University of Canterbury

Master's Thesis

A formal correctness proof of Borůvka's minimum spanning tree algorithm

Author:
Nicolas Robinson-O'Brien

Supervisor:
Walter Guttmann

October 27, 2020

Abstract

Prior work has described an algebraic framework for proving the correctness of Prim's and Kruskal's minimum spanning tree algorithms. We prove partial correctness of an additional minimum spanning tree algorithm, Borůvka's, using the same framework. Our results are formally verified using the automated deduction capabilities of the Isabelle proof assistant. This further demonstrates the suitability of the algebraic framework as a sound abstraction for reasoning about weighted-graph algorithms.

Contents

List of Figures

Chapter 1

Introduction

In 1926, Otakar Borůvka formalized the Minimum Spanning Tree (MST) problem and proposed a solution to it [14]. He was perhaps the first person to do so [34]. Borůvka's MST algorithm computes a minimum spanning tree of a weighted, connected, undirected graph whose edge weights are distinct.

Borůvka's original paper is written in Czech; translations of varying completeness can be found in [34, 63]. The MST problem has since been redefined using the language of graph theory that is more readily understandable, for example, in [20, 78].

Borůvka's MST algorithm has been independently rediscovered by Choquet [18], Florek et al. [25], and Sollin [79]. Some of the work discussed in this thesis is based on the identical algorithms presented by these other authors. However, for simplicity, such work will be referred to under the title of *Borůvka's MST algorithm*.

1.1 A description of Borůvka's MST algorithm

We use standard terminology to describe the MST problem and Borůvka's MST algorithm [75]. Recall that a graph is composed of a set of vertices and a set of edges, where each edge joins two vertices. An edge may have a cost associated with it, called the edge's weight. A graph that contains no cycles is a forest. If there is a sequence of one or more edges between all pairs of vertices in the graph then we say the graph is connected. We call a connected graph that contains no cycles a tree. We call a maximal collection of vertices that are connected in a graph a component. Each component of a forest is a tree.

The MST problem is concerned with finding a subset of the edges of a graph that form a tree, connecting the graph's vertices, where the sum of the weights of the edges is minimal [78]. Since a MST algorithm can find the minimal-cost subset of edges that maintains connectivity, the problem has applications in the design of networks, for example, computer, telecommunication, and transportation networks.

Borůvka's MST algorithm operates as follows. The algorithm takes, as input, an undirected, connected, distinctly-weighted graph. Next, a forest is initialized with n trees, each containing a single vertex, where n is the number of vertices in the graph. While there is more than one tree in that forest, the following step is repeated. For each tree in the forest, find the edge in the graph with the smallest weight among all edges that leave the tree; all edges found in this way are then added to the forest.

By rephrasing this description, it can solve the more general minimum spanning forest problem, that is, to find a subset of the edges of a graph, g, that form a tree for each component of g where the sum of the weights of the edges is minimal. The algorithm starts the same, by initializing a forest to n trees, each containing a single vertex, where n is the number of vertices in the graph. Then, while there are any trees in the forest that could be connected by edges in the graph, the following step is repeated. For each tree that could be connected to another in

the forest by an edge in the graph, find the edge with the smallest weight among all edges that leave the tree; all edges found in this way are then added to the forest.

A contemporary description of the implementation of Borůvka's MST algorithm can be found in [80].

1.2 Significance of Borůvka's MST algorithm

Typically, algorithm textbooks focus on the MST algorithms of Kruskal [54] and Prim [74], such as in [20, 24, 30]. In [78], the authors classify Borůvka's MST algorithm as a less well-known MST algorithm and do not give an implementation. However, Borůvka's MST algorithm is not merely a historic novelty. It has influenced and been the basis for significant improvements in the running-time complexity of MST algorithms.

Often improvements of running-time complexity come by way of the use of different data structures. In 1975, Yao published a modified version of Borůvka's MST algorithm with running-time complexity $O(e \cdot \log \log v)$, where e is the number of edges and v is the number of vertices [84]. Fredman and Tarjan [27] found an implementation using Fibonacci heaps with a running-time complexity of $O(e \cdot \beta(e, v))$, where $\beta(e, v) = \min\{i \mid \log^{(i)} v \leq e/v\}$ is a very slowly-growing function similar to the iterated logarithm function [20]. Their work was improved by Gabow et al. to $O(e \cdot \log \beta(e, v))$ [28]. Our verification will not be concerned with running-time complexity or efficient data structures.

Another promising approach is to make use of parallelism. It is noted that in each iteration of Borůvka's MST algorithm, the selection of an edge for each component of the current forest does not depend on any other edge selection. For this reason, the algorithm is well suited for parallelism. Our verification will be of a sequential version of Borůvka's MST algorithm, not a parallel version.

Chazelle's MST algorithm makes use of Borůvka's MST algorithm and has a running-time complexity of $O(e \cdot \alpha(e, v))$, where α is the inverse of Ackermann's function [16]. This is a nearly linear running-time complexity. It is not known whether a linear-running-time, deterministic MST algorithm exists.

Karger has modified Borůvka's MST algorithm to create a randomized MST algorithm with an expected linear-running-time complexity $O(e)$ [51].

1.3 Formal verification

Formal verification is an approach to reliability assurance for computer software where a higher level of guarantee is given by way of reasoning about a specification and a program that is shown to satisfy that specification. A specification describes what is expected of a program.

Often, the reliability of algorithms is managed by manual or automated tests. It is not possible for such an empirical method of reliability testing to provide verification that any but the simplest algorithms are correct. Rather, a failed test will verify that a program is incorrect. Instead, a formal mathematical proof may be used to verify that an algorithm meets its specification, that is, the algorithm will produce the correct output for all possible inputs. The correctness of an algorithm can be formally proved, in the sense that it meets its specification, by expressing the semantics of the algorithm. This amounts to expressing the algorithm and the specification mathematically, and then using logic to show that the mathematical expression of the algorithm satisfies the mathematical expression of the specification.

The associated problem of manual proof-verification may be somewhat alleviated by the use of a program, such as a proof assistant, that is capable of mechanically verifying proofs.

1.3.1 Proof assistants

There are many proof assistants available. See, for example, the survey [12] and book [83]. The purpose of such proof assistants is to automatically verify mathematical proofs and in some cases assist with proof finding. Some of the more prominent proof assistants include Coq [7], *HOL* [32], Mizar [62], and Isabelle/HOL [69]. All of these proof assistants include proof libraries of formally verified mathematics, for example, sets, relations, natural numbers, and integers.

Coq is an interactive proof assistant that facilitates the production of specifications and programs which are proved to be consistent with those specifications. It implements a mathematical language Gallina, based on the Calculus of Inductive Constructions, that combines higher-order logic and a richly-typed functional programming language. Coq differs from many other proof assistants in that it provides a mechanism to extract a working program from the proof. There are plugins that enhance Coq with automated proof-finding tools, for example, SMTCoq [23].

HOL is an interactive proof assistant for the form of predicate logic by the same name. It is released with built-in proof-finding tools and a mechanical verification system. To avoid confusion we will not make any further references to this proof assistant and in the remainder of this thesis, Higher Order Logic (HOL) should be taken to mean the form of mathematical predicate logic.

Mizar consists of an input language for the formalization of proofs and a mechanical verification tool. The language is close in appearance to mathematics, and therefore lends itself to be more readily understood by a person schooled in mathematics but unfamiliar with Mizar.

Isabelle/HOL has similar features. It allows both manual proof-writing as well as providing proof-finding tools and allowing the use of external proof-finding tools. It has a proof verification system and an expressive language similar to mathematics. It is most sensible for us to use Isabelle/HOL due to the extensive related work that has already been done in this system.

1.3.2 Isabelle/HOL

We chose to use Isabelle/HOL to produce the formal verification of Borůvka's MST algorithm. This was primarily because the algebraic framework that our work builds on and a substantial number of relevant lemmas had already been published in the Archive of Formal Proofs, a repository of proofs that have been verified by Isabelle/HOL.

Our Isabelle/HOL theory artifact begins by inheriting useful library files, including a library for Hoare logic proofs and the theory files that include the definition of the algebraic framework that our proof is based in. We work in m-k-Stone-Kleene relation algebras. This is an algebra based on m-Kleene algebras, discussed in Section 2.3.6.

Isabelle/HOL provides utilities that simplify proof development. A list of lemmas is presented in a separate pane. Fast navigation hotkeys are available to jump to key definitions and navigation history is maintained so that a user is able to return to their previous editing positions. The proof state at the cursor position, in particular the proof subgoal, is also presented in a separate pane.

We have relied heavily on Sledgehammer, a proof-finding tool that is integrated with Isabelle/HOL [73]. Sledgehammer attempts to automatically find proofs for goals by making requests to various automatic theorem provers. The proofs that are found are then verified by Isabelle/HOL. Sledgehammer has heuristics to select relevant lemmas that are available in the scope of the proof goal. These lemmas are provided to the automatic theorem provers. In order to discharge proof goals without this tool, the author would need to know the name of the lemmas required that prove each goal. This is a considerable ask, especially for an author who is unfamiliar with the available lemmas that may be spread out over many proof files. While Sledgehammer is not typically able to find proofs for complex goals, it has been a valuable addition.

Finally, we have used a Hoare-logic verification generator library that comes with Isabelle/HOL [65, 66]. This library has allowed us to input our formalized algorithm, from Section 3.2, and have a list of proof goals generated that, once satisfied, imply that the algorithm is correct. We show partial correctness. This means that whenever the input satisfies the precondition and the algorithm terminates, its output satisfies the postcondition. We discuss the work required to prove termination in Chapter 5.

1.4 Aim of this thesis

The aim of our research is to provide a machine-verified formal partial-correctness proof for Borůvka's MST algorithm using Isabelle/HOL. To our knowledge, there is no formal proof of correctness for this algorithm, machine-verified or otherwise.

Guttmann has recently introduced new algebras [44] that have proved useful for reasoning about weighted-graph algorithms. In particular, they have been used to complete total-correctness proofs of both Prim's and Kruskal's MST algorithms. We intend to use the same algebras to further demonstrate their suitability as a framework for reasoning about weighted graphs in general and constructing proofs for MST algorithms in particular.

1.5 Organization of this thesis

In Chapter 2 we discuss mathematical structures that we use in our work.

In Section 2.1 we give definitions for various graph structures and concepts that we use throughout the paper. Almost all of these are standard in the literature. Of particular importance is the *rooted directed forest*, a structure that we use often in our proof.

In Section 2.2 we discuss minimum spanning trees in general. We give a definition of Borůvka's MST problem in Section 2.2.2 as well as an example of its operation. Some notable differences to Kruskal's MST algorithm are also mentioned.

In Section 2.3 we discuss binary relations. There is a straightforward model of a directed graph as a relation. Binary relations provide a good base to reason about unweighted graphs and, with some changes to the algebraic structure, weighted graphs. Our proof is based on an existing algebraic framework and we give definitions for these structures and discuss ideas that are used in the remainder of the thesis.

In Chapter 3 we introduce the algebras that our formalization and proof are completed in as well as present and describe our formalization of Borůvka's MST algorithm.

In Section 3.1, we introduce m-k-Stone-Kleene relation algebras, that our proof is completed in. An m-k-Stone-Kleene relation algebra combines m-Kleene relation algebras, discussed in Section 2.3.6, the Tarski rule and a new binary operation, k, that models the selection of a component in a graph.

In Section 3.2 we present our formalization of Borůvka's MST algorithm. We make three changes to the algorithm as it is described in Section 2.2.2. The first is that our formalization uses a variable to track a forest. As the algorithm progresses, this forest grows to be the MST. The second is that the formalization solves the minimum spanning forest problem. In the case where the input graph is connected the output forest will be a tree. Lastly, we do not require that the input graph's edge weights are distinct. Instead, we predicate the addition of an edge to the forest variable on the condition that such an addition will not create a cycle.

In this thesis, we include the Isabelle/HOL theory in Appendix B. We do not discuss all the theorems contained in it, though we do discuss a selection of them in Chapter 4.

In Section 4.2 we introduce a new abstraction, E-forests, that is used in our proof to reason about reachability in the remainder of Chapter 4.

In Section 4.3 we formally specify what it means to be a minimum spanning forest, that is, what the output of our formalization should satisfy. Additionally, we give the invariants used

to show that the specification is met.

In Section 4.4 we discuss how our invariants are established and maintained. We provide several examples, in various levels of detail.

1.6 Contributions

The main contributions of our work are:

- A new algebraic structure, k-Stone relation algebras, that extends Stone relation algebras with a component selection operation, k. The k operation models selection of components in graphs. We also introduce m-k-Stone-Kleene relation algebras, an algebraic structure that combines k-Stone relation algebras, m-Kleene algebras and the Tarski rule. Our formalization of Borůvka's MST algorithm and its partial-correctness proof is done in these algebras. These contributions are presented in Section 3.1.
- A formal specification of Borůvka's MST algorithm, presented in Section 3.2.
- A new abstraction, E-forests, that we use to model reachability, presented in Section 4.2.
- A Hoare-logic formally-verified partial-correctness proof of Borůvka's MST algorithm, discussed in Section 4.4.
- Formal verification of our proof as Isabelle/HOL theorems, included in Appendix B.

1.7 Related work

A formal proof reasons about correctness using a step-by-step argument expressed in a formal language such as mathematics, while an informal proof does not. The correctness of graph algorithms in general, and MST algorithms in particular, have typically been argued informally, for example, in [20, 78]. Borůvka included such an informal proof of his MST algorithm in his original work [14].

Relation algebras provide a framework to reason about and develop unweighted-graph algorithms. An unweighted graph has a direct representation as a Boolean matrix and hence a binary relation. Such relations may represent both directed and undirected graphs. It is therefore not surprising that relation algebras have been used to reason about unweighted graph algorithms with some success.

In [52], the authors use relation algebras to improve computation performance for the minimum vertex cover, maximum clique, and maximum independent set problems. In [5], the authors present a relation-algebraic approach to the tournament choice problem. This involves the computation of choice sets using relation algebra and RelView [3, 4], a software system that assists with computation on Boolean matrices.

In [6], the authors use relation algebras to compute spanning trees of undirected graphs. A proof is also given for a variant of Prim's MST algorithm using relation algebras. The proof includes arguments about the Galois connection between incidence relations and adjacency relations. While the incidence relations make computation more difficult, they provide a more convenient base to generalize with weighted graphs. No automated proof is provided. However, an implementation of the algorithm is given in RelView. The authors discuss how the use of a formal calculus that allows automated verification would be more ideal than the informal approach taken.

Relation algebras are not so suitable for reasoning about weighted graphs, so some authors have looked to other algebraic frameworks for this purpose. Semirings have been applied to path finding problems in graphs [31, 49]. Mohri discusses the application of semirings to weighted-graph shortest-path problems in [61]. The author notes that other authors have also investigated

the use of semiring frameworks with respect to the all-pairs shortest-distance problem. A formal proof is given, though it is not machine-verified.

A general algebraic framework for the MST problem is introduced in [9]. It leverages constraint-based semirings and is inspired by the work of Mohri. The paper includes a proof of a variant of Kruskal's MST algorithm. The proof is informal and not automated.

In [68], the authors survey recent work formally verifying algorithms from the algorithm textbook by Cormen et al. [20].

Use has been made of theorem provers to construct machine-verified proofs for some MST algorithms. A distributed MST algorithm by Gallagher et al. [29] was formally proved correct by Hesselink [48] in the Nqthm framework [15]. Abrial et al. use the B event-based method in the Atelier B environment to show the correctness of Prim's MST algorithm [1].

A proof, machine-verified by Referee [71], is presented in [72] showing that a connected claw-free graph has a Hamiltonian cycle in its square and, if it has an even number of vertices, owns a perfect matching.

A formal verification has been constructed in [56] for Dijkstra's single source shortest path algorithm, Prim's MST algorithm, and the Ford-Fulkerson maximum network flow algorithm. It is a machine-verified proof written in Mizar [62].

A library for proving various properties of graph theory is presented in [22]. The library is written in Coq [7] and is used to formally verify: Menger's theorem, the excluded-minor characterization of treewidth-two graphs, and a correspondence between multigraphs of treewidth at most two and terms of certain algebras. A formal verification, also written in Coq, of the iterated register coalescing algorithm is presented in [10]. This paper includes a proof that the algorithm terminates.

A graph library [70] has also been constructed for Isabelle/HOL [69]. It covers simple graphs and multigraphs. A formalization of planar graphs was constructed as part of the Flyspeck project [67] in Isabelle/HOL.

A formal verification is given in Isabelle/HOL for two network flow algorithms, Edmonds-Karp and generic push–relabel, in [55]. The time complexity bounds, $O(VE^2)$ and $O(V^2E)$ respectively, are also formally verified in that paper.

Guttmann presents a Stone-Kleene relation-algebraic framework, in [43], and uses it to formally verify Prim's MST algorithm. The algebra permits instantiation by weighted matrices. He extends this work in [44] including a proof of Kruskal's MST algorithm. This is a proof of total correctness, that is, the algorithm is shown to terminate. Use is made of Isabelle/HOL and its integrated, automated theorem provers to mechanically verify these results. The relevant Isabelle/HOL theorems, in particular those of Stone relation algebras [40], Stone-Kleene relation algebras [39], and aggregation algebras [41], are included in the Archive of Formal Proofs.

Our work is based on the work by Guttmann and we have been able to reuse many results from his work. The verification of both Prim's and Kruskal's MST algorithms use Hoare logic and the majority of both proofs are performed in Stone-Kleene relation algebras. The proof of Kruskal's MST algorithm is particularly related to our work because, similar to Borůvka's MST algorithm, it is concerned with growing a forest whereas Prim's MST algorithm grows a tree. This is discussed further in Section 2.2.1.

Guttmann gives different formal specifications of the MST problem for Prim's and Kruskal's MST algorithms. The specification for Prim's MST algorithm outputs a tree whereas the specification for Kruskal's MST algorithm outputs a forest. Guttmann notes that it should be possible to modify the proof of Prim's MST algorithm so that it uses the same specification as for Kruskal's MST algorithm since trees are a special case of forests.

We use the minimum spanning forest specification given for Kruskal's MST algorithm since a forest structure is more suitable for our work. This specification is discussed in Section 4.3.1. The MST problem is defined at the start of Section 2.2.

Chapter 2

Background

Many mathematical structures are used in this thesis. In this chapter, we give definitions for these structures and discuss concepts that will be used. We introduce terminology that is used to discuss the operation of Borůvka's MST algorithm as well as our correctness proof. In particular, we describe graphs, MSTs, relations, orders, lattices, and Stone algebras.

2.1 Graphs

We start with graphs. Many textbooks are available that further discuss properties of graphs [13, 26, 30, 33, 46, 75, 82]. The definitions we use come from [26]. Note that for our applications we consider only finite graphs.

Though it is possible to have graphs comprised of both directed and undirected parts, in this thesis, when referring to a graph, we will either be talking about an undirected or a directed graph. When introducing MST algorithms in Section 2.2, we will be talking exclusively about undirected graphs. However, our proof is done with directed graphs.

2.1.1 Undirected graphs

An *undirected graph* (graph), $G = \langle V\,,\ E\rangle$, is a pair of sets, V and E, where V is non-empty and finite and E may be empty and is also finite. V is the set of *vertices* in G and E is the set of *edges* in G. Each edge has a set of one or two vertices associated with it that are called its *endpoints*.

A *loop* is an edge that joins a single vertex to itself. Therefore, a loop has only one endpoint. For example in Figure 2.1(a), vertex a is the endpoint of a loop. A *multi-edge* is a collection of two or more edges that have identical endpoints. For example, vertices e and d share two edges in Figure 2.1(a). We do not deal with graphs containing multi-edges, so every edge can be uniquely identified by its endpoints.

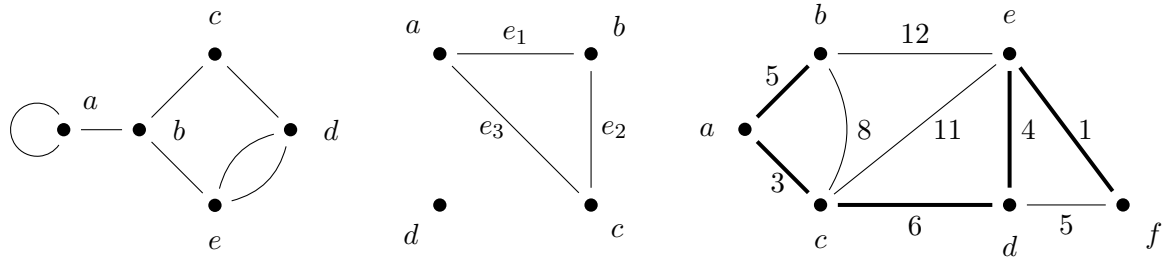
An edge is said to be *incident* to a vertex that it connects to and two vertices are called *adjacent* if they are connected by an edge. For example, in Figure 2.1(b), edge e_3 is incident to vertices a and c while vertex d is adjacent to no vertices.

A *walk* in a graph is an alternating sequence of vertices and edges $(v_0, e_1, v_1, e_2, \ldots, e_{n-1}, v_{n-1}, e_n, v_n)$, such that for $i = 1, \ldots, n$, the vertices v_{i-1} and v_i are the endpoints of the edge e_i. A walk is *closed* if v_0 is the same as v_n, otherwise, it is *open*.

A *trail* is a walk where no edge occurs more than once. A *path* is a trail, $(v_0, e_1, v_1, e_2, \ldots, e_{n-1}, v_{n-1}, e_n, v_n)$, where the vertices are distinct, except that v_0 may be the same as v_n.

A *cycle* is a closed path that has at least one edge. If a graph contains no cycles, then it is called *acyclic*.

For example, Figure 2.1(b) shows a walk, $(a, e_1, b, e_2, c, e_3, a)$. This is also a trail, a path, and a cycle. Figure 2.1(c) depicts a path in bold.



(a) A graph with a loop and multi-edge.

(b) A graph with two components.

(c) A weighted graph with a path marked in bold.

Figure 2.1: Examples of undirected graphs.

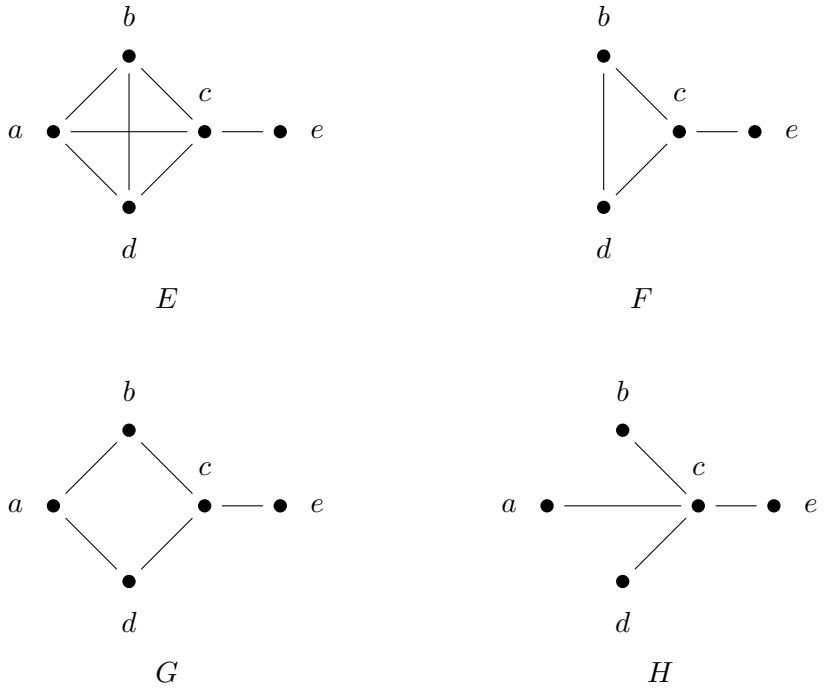


Figure 2.2: A graph, E, and three of its subgraphs F, G, and H.

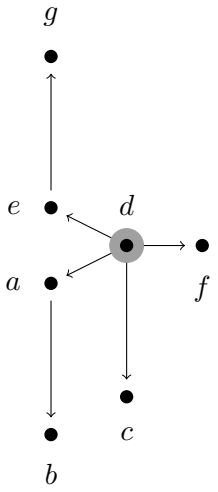
A graph is *connected* if, for all pairs of vertices $u, v \in V$, there is a walk from u to v. For example, the graphs in Figures 2.1(a) and 2.1(c) are connected while Figure 2.1(b) depicts a graph that is not connected; there is no path from d to vertices a, b and c. A maximal collection of vertices that are connected in a graph we call a *component* of the graph so that Figure 2.1(b) has two components.

A *forest* is an acyclic graph and a *tree* is a connected forest.

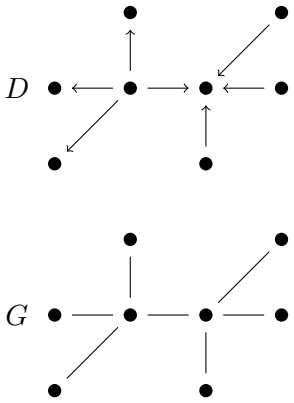
A *subgraph*, $G' = \langle V' , E' \rangle$, is a subset of the vertices and edges of a graph, G, such that $V' \subseteq V$, $E' \subseteq E$, and any edge in E' has its endpoints in V'. A *spanning subgraph* of G is a subgraph that contains all the vertices of G. A *spanning tree* is a spanning subgraph that is a tree. For example, in Figure 2.2, F is a subgraph of E but is not a spanning subgraph. G is a spanning subgraph of E but is not a tree. H is a spanning tree of E.

Vertices and edges may be *colored*. This is when a color is assigned to a vertex or edge, often to partition a graph into subgraphs.

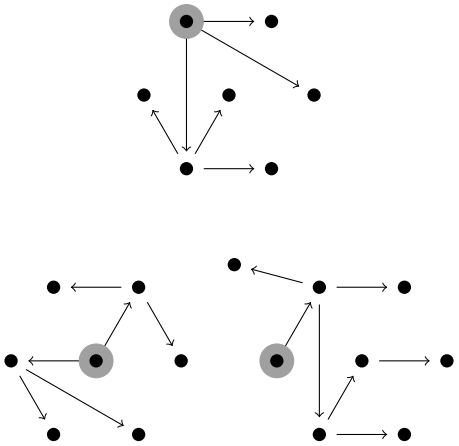
A *weighted graph*, $G = \langle V , E , w \rangle$, is a graph where w is a function, $w : E \mapsto \mathbb{R}$, that maps edges to real values. Figure 2.1(c) is such a weighted graph with, for example, $w(\{a, b\}) = 5$ and $w(\{e, f\}) = 1$.



(a) An example digraph, also a rooted directed tree, with the root highlighted gray.



(b) An example digraph, D, and its underlying graph, G.



(c) A rooted directed forest comprised of three rooted directed trees. The roots of the rooted directed trees are highlighted gray.

Figure 2.3: Examples of digraphs.

2.1.2 Directed graphs

A *directed graph* (digraph), $D = \langle V\,,\ A\rangle$, is a pair of sets, V and A. The set of vertices, V, is non-empty and finite. The set of directed edges, A, is finite where a directed edge is an ordered pair of vertices, $(u, v) \in V \times V$. We also call a directed edge an *arc*.

Vertices in a digraph are also called adjacent when connected by an arc, and an arc is incident to the vertices it connects to.

Let (x, y) be an arc of a digraph. The order of this pair is to be interpreted as the arc originating at vertex x (the *source*) and terminating at vertex y (the *target*). We depict the arcs of a digraph with an arrow pointing to the target. For example, the graph in Figure 2.3(a) has arcs $\{(a, b), (d, a), (d, c), (d, e), (d, f), (e, g)\}$.

A *directed walk* in a digraph is an alternating sequence of vertices and arcs $(v_0, a_1, v_1, a_2, \ldots, a_{n-1}, v_{n-1}, a_n, v_n)$, such that for $i = 1, \ldots, n$ the source of a_i is v_{i-1} and the target of a_i is v_i. A directed walk is *closed* if v_0 is the same as v_n, otherwise, it is *open*. Then, similar to undirected graphs, a *directed trail* is a directed walk where no arc occurs more than once, a *directed path* is a directed trail where the vertices are distinct, except that v_0 may be the same as v_n, and a *directed cycle* is a closed directed path that has at least one arc.

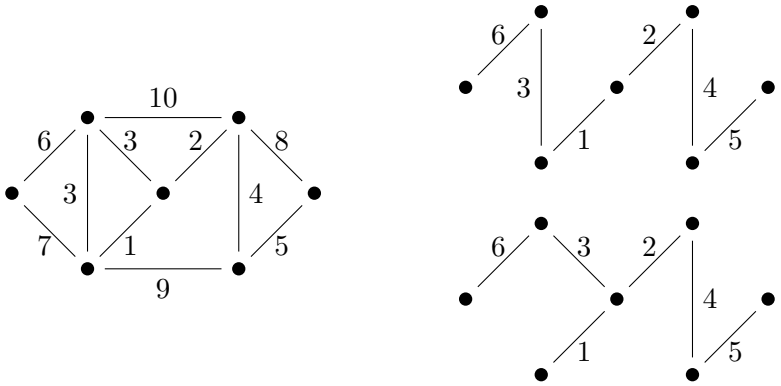


Figure 2.4: A graph and its two minimum spanning trees.

An *acyclic digraph* is a digraph that has no directed cycles.

The *underlying graph* of a digraph is the undirected graph that results if all direction information is removed from the arcs. In the case where a digraph, D, has arcs in both directions between two vertices, for example, (u, v) and (v, u), the underlying graph of D has only one edge between u and v, not a multi-edge.

A digraph is connected if its underlying graph is connected. A component of a digraph is a maximal collection of vertices that are connected.

A *directed tree* is an acyclic digraph whose underlying graph is a tree. For example, the underlying graph, G, of a digraph, D, is shown in Figure 2.3(b). Since G is a tree D is a directed tree.

A *rooted directed tree* is a directed tree that has a vertex r, the *root*, such that for all other vertices v there is a directed path from r to v in the directed tree. This variation, where all the arcs are directed away from the root, is also known as an arborescence. The variant of a rooted directed tree that has all arcs directed towards the root is also known as an anti-arborescence. For example, Figure 2.3(a) depicts a rooted directed tree, in particular, an arborescence. Neither graph D nor G in Figure 2.3(b) is a rooted directed tree.

A *rooted directed forest* is a digraph where each component is a rooted directed tree. For example, Figure 2.3(c) shows three rooted directed trees that together form a rooted directed forest. This structure is acyclic and each vertex is the target of at most one arc. The arcs form directed paths away from the roots. Another representation may be had if all the arc directions are reversed. In this case, each vertex would be the source of at most one arc and the arcs would form directed paths towards the roots. A rooted directed forest is a generalization of a rooted directed tree. While the graph in Figure 2.3(a) is a rooted directed tree, it is also a rooted directed forest.

2.2 Minimum spanning trees

A *minimum-weighted spanning tree* of an undirected, connected, weighted graph, $G = \langle V\,,\ E\,,\ w\rangle$, is a spanning tree, $T = \langle V\,,\ E'\rangle$, of G where the sum of the weights of the edges, E', of T

$$w(T) = \sum_{\{u,v\}\in E'} w(\{u, v\})$$

is minimal, that is, less than or equal to the weight of any other spanning tree of G.

We assume it is understood that we are minimizing weight, so we simply refer to a MST. The graph in Figure 2.4 has two MSTs, each with total weight 21.

The MST problem is to find a MST of a graph. The search for solutions to the MST problem has a rich history [34, 60, 64]. There are many obvious applications of solutions to the

MST problem including development of computer, communication, transportation, and other networks.

2.2.1 Prim's and Kruskal's MST algorithms

Descriptions of the common algorithms that find a MST can be found in [20, 24, 30, 60, 78, 80]. Two well-known algorithms that solve the MST problem are those by Prim [74] and Kruskal [54].

Prim's MST algorithm

Prim's MST algorithm operates as follows. Given an undirected, connected, weighted graph, G, as input, initialize a tree, T, with an arbitrary vertex from G. While there are still vertices in G that are not in T repeat the following:

1. Determine vertices in G that are not in T, but are adjacent to a vertex in T. Select from those a vertex, v, connected to T by an edge with minimal weight, e.
2. Add v and e to T.

When the while-condition fails the algorithm outputs T. After each iteration of the while-loop, T is a tree. We call this an *invariant*.

Kruskal's MST algorithm

Kruskal's MST algorithm operates as follows. Take as input an undirected, connected, weighted graph, G. Initialize a forest, F, with one component for each vertex of G and no edges. For each edge, e, in G, ordered by weight in non-decreasing order, repeat the following step:

1. Test whether adding e to F creates a cycle. If it does not then add e to F.

When the while-condition fails the algorithm outputs F. An invariant of the while-loop is that F is a forest.

There are notable differences between these algorithms. Prim's MST algorithm enumerates the vertices of a graph, while Kruskal's MST algorithm enumerates the edges of a graph. In the form presented above, if the input to Prim's MST algorithm is not a connected graph then there will be undefined behavior. This is because at some stage there will be a vertex in G that is not in T but is not adjacent to any vertex in T so the first step of the loop cannot be performed. Kruskal's MST algorithm does have well-defined behavior for an input graph that is not connected. In this case, the output will be a forest where each tree of the forest is a MST. This is called a *minimum spanning forest*.

2.2.2 Borůvka's MST algorithm

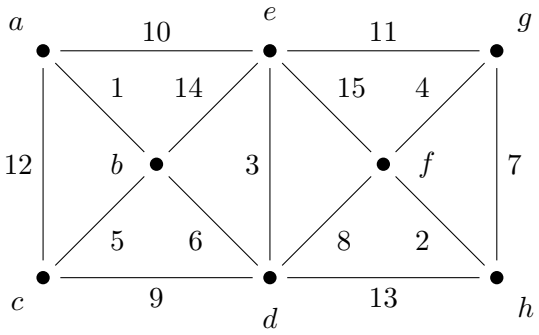
Borůvka's MST algorithm operates as follows. Given an undirected, connected, distinctly-weighted graph, color all vertices gray and assign them to a forest, F, with no edges. Repeat the following step until only one gray tree remains.

For every gray tree, T, in F:

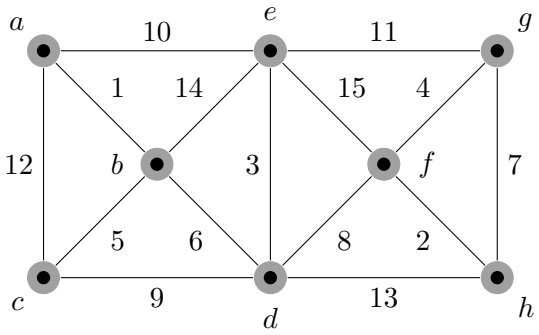
1. Determine those edges incident to vertices in T that do not have both endpoints in T.
2. Select from those edges the one with minimum weight and color it gray.

When the algorithm terminates, it outputs those colored edges and vertices.

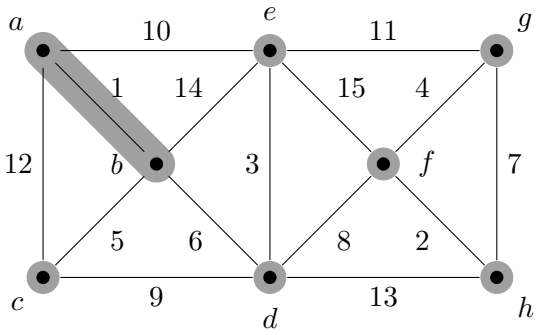
This algorithm contains two loops. The outer loop continues while F is not connected and the inner loop iterates over the trees of F. Coloring an edge in step 2 does not change the trees until the current iteration of the outer loop completes.



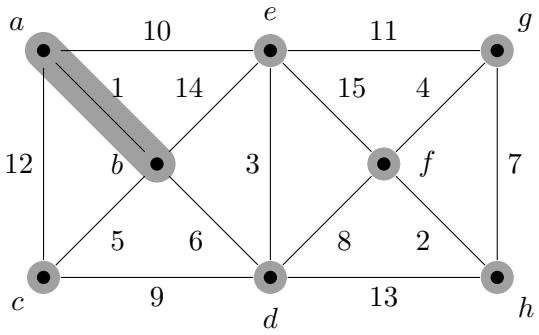
(a-1) The algorithm accepts as input an undirected, connected, distinctly-weighted graph.



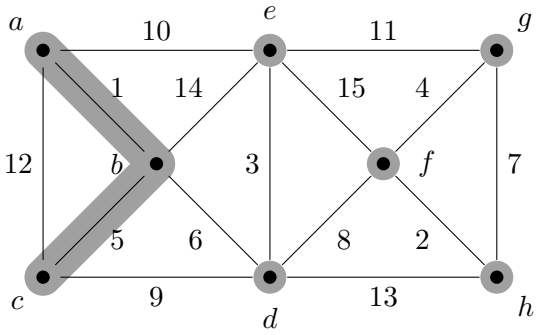
(a-2) Each vertex in the graph is colored gray, initializing n trivial trees.



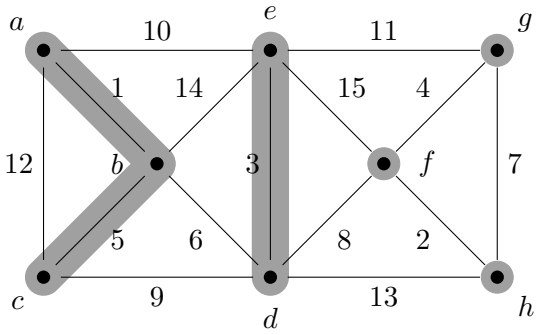
(b-1) The tree with vertex a is processed, having an incident edge with minimal weight of 1.



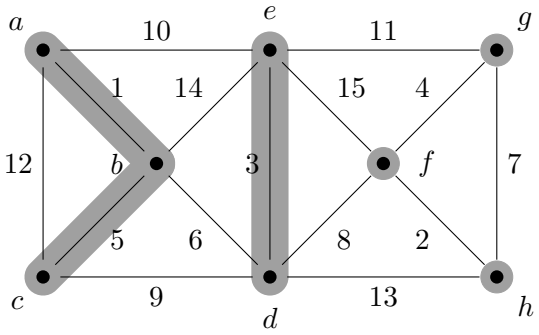
(b-2) The tree with vertex b is processed, having an incident edge with minimal weight of 1.



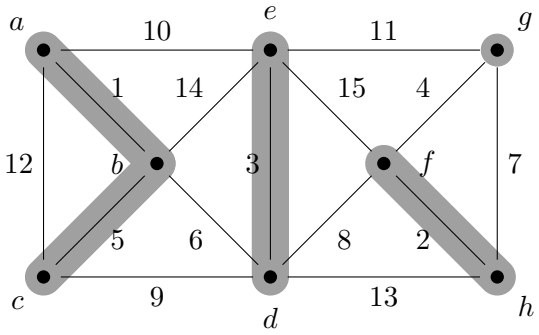
(b-3) The tree with vertex c is processed, having an incident edge with minimal weight of 5.



(b-4) The tree with vertex d is processed, having an incident edge with minimal weight of 3.

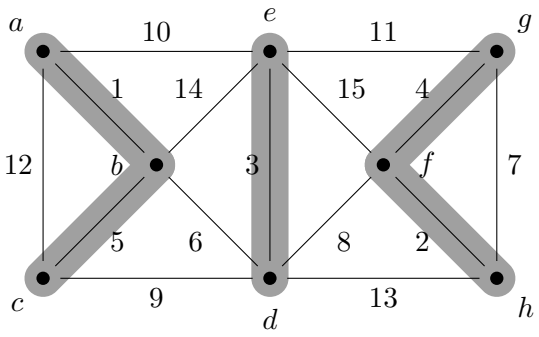


(b-5) The tree with vertex e is processed, having an incident edge with minimal weight of 3.

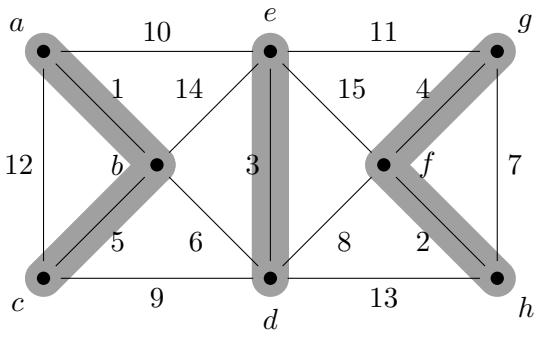


(b-6) The tree with vertex f is processed, having an incident edge with minimal weight of 2.

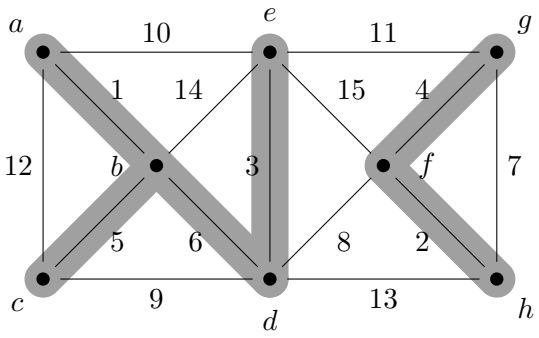
Figure 2.5: The operation of Borůvka's MST algorithm.



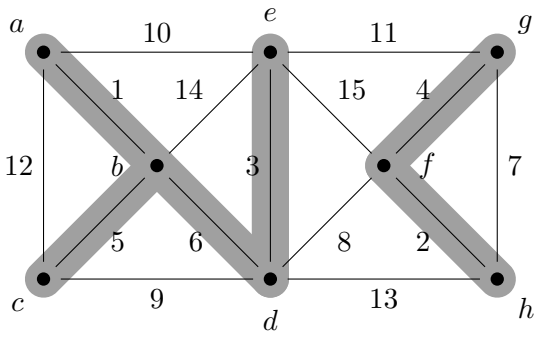
(b-7) The tree with vertex g is processed, having an incident edge with minimal weight of 4.



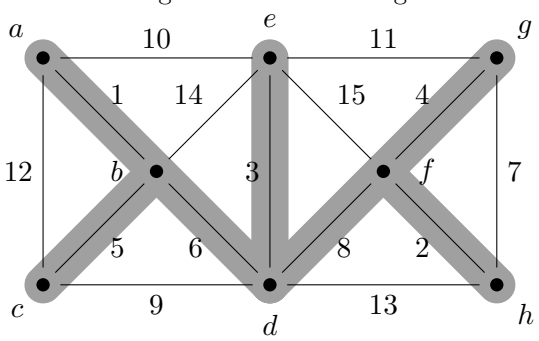
(b-8) The tree with vertex h is processed, having an incident edge with minimal weight of 2.



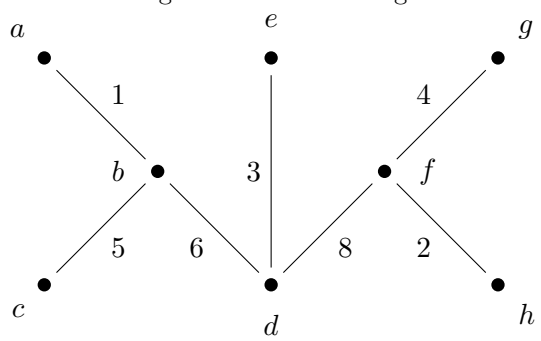
(c-1) The left tree (with vertices a, b and c) has an incident edge with minimal weight 6.



(c-2) The center tree (with vertices d and e) has an incident edge with minimal weight 6.



(c-3) The right tree (with vertices f, g, and h) has an incident edge with minimal weight 8.



(d) Only one tree remains and the algorithm outputs the MST.

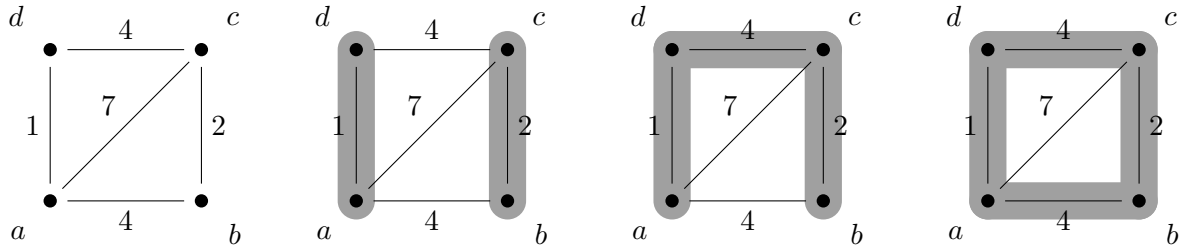
Figure 2.5 (continued): The operation of Borůvka's MST algorithm.

Example of operation

An example of the operation of this algorithm is given in Figure 2.5. The state of the forest, F is indicated with gray coloring. The initialization steps are performed in Figures 2.5(a-1) and 2.5(a-2). At this stage, the loops have not started.

There are two iterations of the outer loop. The sub-figures that depict the operations that occur in the first iteration of the outer loop are prefixed with (b). The sub-figures that depict the operations that occur in the second iteration of the outer loop are prefixed with (c). For example, Figure 2.5(b-2) shows the outcome of the second iteration of the inner loop in the first iteration of the outer loop and Figure 2.5(c-1) shows the outcome of the first iteration of the inner loop in the second iteration of the outer loop. While F is updated in each iteration of the inner loop, the trees that are being processed do not change until the end of each iteration of the outer loop. For this reason, the eight trees shown in Figure 2.5(a-2) are processed in Figures 2.5(b-1) to 2.5(b-8) irrespective of the edges being colored in F.

In Figure 2.5(b-1), the outer loop has been entered for the first time and a single iteration



(a) An input graph with non-distinct edge weights.

(b) After one iteration of the outer loop, the forest contains two trees.

(c-1) The edge between d and c may be added next.

(c-2) Adding the edge between a and b creates a cycle.

Figure 2.6: Input graphs to Borůvka's MST algorithm need to have distinct edge weights.

of the inner loop has been performed to process the tree with vertex a, adding the edge with weight 1 to F. In Figure 2.5(b-2) the second iteration of the inner loop has processed the tree with vertex b. The minimum-weight incident edge to this tree, the edge with weight 1, is already in F so this iteration of the inner loop makes no observable changes to F. The remaining trees in F are processed in the same manner, as shown in Figures 2.5(b-3) to 2.5(b-8).

Figure 2.5(b-8) shows the state of F as the first iteration of the outer loop has completed.

The second iteration of the outer loop has three trees to process. In Figure 2.5(c-1) the tree in F containing vertices a, b and c is processed first and the edge with weight 6 is added to F. The same edge is the minimum-weight incident edge to the tree with vertices d and e so that there is no observable change to F in the second iteration of the inner loop. This is shown in Figure 2.5(c-2). The edge with weight 8 is added to F when the tree with vertices f, g, and h is processed, as shown in Figure 2.5(c-3).

The inner loop then exits as each tree in F has been processed. The outer loop also exits, as F is connected, and the MST shown in Figure 2.5(d) is output. This example required two iterations of the outer loop to complete.

Distinct edge weights

Unlike Kruskal's and Prim's MST algorithms, Borůvka's MST algorithm requires the input edge weights to be distinct. The reason for this may be seen if we trace an example for a graph that has two edges with the same weight.

Consider the input graph in Figure 2.6(a). There are two edges with weight 4. The algorithm will proceed as described above until it reaches a situation where there are two trees in F, comprised of vertex pairs $\{a, d\}$ and $\{b, c\}$, as shown in Figure 2.6(b). Now the algorithm proceeds for each tree, T, in F to select the minimum-weighted edge that does not have both endpoints in T. Because the edge weights are not distinct, different edges may be chosen. For the tree with vertices $\{a, d\}$, it may select the edge between d and c. The outcome of this selection is shown in Figure 2.6(c-1). When the algorithm processes the tree with vertices $\{b, c\}$ there are two edges it may choose. If the edge between vertices a and b is added then a cycle is created so that F will no longer be a forest. This selection is shown in Figure 2.6(c-2).

2.3 Algebras for reasoning about graphs

We are interested in mathematical representations of graphs that will be useful for reasoning about algorithms that operate on graphs in general and Borůvka's MST algorithm in particular. Binary relations provide a good base to reason about unweighted graphs, but because weighted graphs do not have a direct representation as a binary matrix, we require some alternative structure. Guttmann [38] proposes such a suitable structure. Weighted graphs, represented as

$$\mathsf{L} = \begin{array}{c} \\ x \\ y \\ z \end{array} \begin{array}{c} \begin{array}{ccc} x & y & z \end{array} \\ \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix} \end{array} \qquad \mathsf{O} = \begin{array}{c} \\ x \\ y \\ z \end{array} \begin{array}{c} \begin{array}{ccc} x & y & z \end{array} \\ \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 2.7: The universal, L, and empty, O, relations.

matrices whose entries range over the real numbers (denoting the weight of edges) extended with two elements (denoting no edge and an edge of unknown weight) are an instance of this structure. In this section, we discuss the relevant algebras.

2.3.1 Relations

Relations and their algebraic structure are discussed in more detail in [59, 77].

A *binary relation* on a non-empty set S is a subset of the Cartesian product $S \times S = \{(x, y) \mid x, y \in S\}$. For $x, y \in S$ we say that if $(x, y) \in R$ then x is related to y by R. For example, the less-than relation (of infinite size) on $\mathbb{N}$ is

$$\begin{aligned} \{&(0,1),(0,2),(0,3),\ldots,\\ &(1,2),(1,3),(1,4),\ldots,\\ &(2,3),(2,4),(2,5),\ldots,\\ &\ldots\} \end{aligned}$$

There is a straightforward representation of a digraph as a binary relation, R, over the set of vertices. Such a representation is an adjacency matrix where a 1 entry in row i and column j, $(i, j) \in R$, denotes the existence of an arc with source i and target j in the graph. Unweighted, undirected graphs may also be represented using binary relations. This could be done by having $(i, j) \in R$ and $(j, i) \in R$. Such a pair of entries would be interpreted as an undirected edge between vertices i and j.

The *universal relation*, L, over a set, S, is all pairs of the Cartesian product

$$\mathsf{L} = S \times S = \{(x, y) \mid x, y \in S\}$$

The *empty relation*, O, on set S, is the empty set, $\mathsf{O} = \emptyset$.

The interpretation of these relations for a graph with vertex set, V, is all arcs, $E = V \times V$, and no arcs respectively. For example, Figure 2.7 depicts how these would look for a graph of three vertices.

Relations are sets and many operations of sets apply to relations, such as *complement*, *union*, *intersection*, and *subset*. Given a relation, R, on a set S, the complement of R, $\overline{R}$, is the set of all pairs of elements not in R but where each element of the pair is in S

$$\overline{R} = \{(x, y) \mid (x, y) \notin R \wedge x, y \in S\}$$

Then, for example, the *involutive* property of complement holds for relations so that $\overline{\overline{R}} = R$. Consider also the union and intersection of two relations, Q and R, denoted as $Q \cup R$ and $Q \cap R$

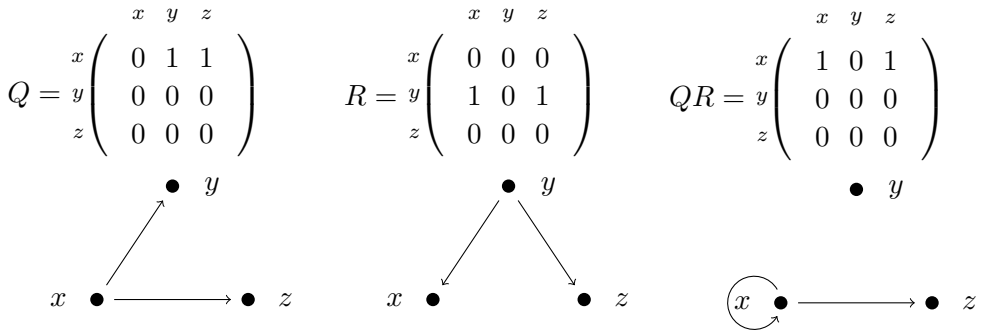


Figure 2.8: An interpretation of relation composition for digraphs.

respectively, where

$$Q \cup R = \{(x, y) \mid (x, y) \in Q \vee (x, y) \in R\}$$
$$Q \cap R = \{(x, y) \mid (x, y) \in Q \wedge (x, y) \in R\}$$

Then, for example, both union and intersection of relations are *associative*, $P \cup (Q \cup R) = (P \cup Q) \cup R$ and $P \cap (Q \cap R) = (P \cap Q) \cap R$; *commutative*, $Q \cup R = R \cup Q$ and $Q \cap R = R \cap Q$; and *idempotent*, $R \cup R = R$ and $R \cap R = R$. The subset relationship, $R \subseteq Q$, is defined as

$$\forall x, y \in S \colon (x, y) \in R \Rightarrow (x, y) \in Q$$

Other operations are defined for relations. The composition of two relations, Q and R, is denoted as QR and defined as

$$QR = \{(x, y) \mid \exists z : (x, z) \in Q \wedge (z, y) \in R\}$$

An interpretation of this for a digraph is a relation, QR, representing arcs such that, if we were to step from their source to their target, we could reach exactly those vertices if we had taken consecutive steps in Q and then R. For example, in Figure 2.8 the arc from x to itself is in QR because we can step from vertex x to y in Q and then from y to x in R.

The *identity relation*, I, on a set S is

$$\mathsf{I} = \{(x, x) \mid x \in S\}$$

A graph interpretation of the identity relation is all loops of the vertices. The identity relation is the unit of composition for relations, so $R\mathsf{I} = R = \mathsf{I}R$.

Relations may be composed with themselves and there is an exponent notation for this, for example, $R^2 = RR$. More generally, the i-th power of a relation, R^i, is defined as

$$R^i = \begin{cases} \mathsf{I} & i = 0 \\ RR^{i-1} & i \geq 1 \end{cases}$$

A relation is said to be *transitive* if the product $R^2 \subseteq R$. The *transitive closure* of R is denoted R^+ and defined as

$$R^+ = \bigcup_{i \geq 1} R^i = R^1 \cup R^2 \cup R^3 \cup \ldots$$

A relation, R, is said to be *reflexive* if $\mathsf{I} \subseteq R$, *co-reflexive* if $R \subseteq \mathsf{I}$, and *irreflexive* if $R \subseteq \bar{\mathsf{I}}$. The *reflexive-transitive closure* of a relation, R, is denoted R^* and defined as

$$R^* = \mathsf{I} \cup R^+$$

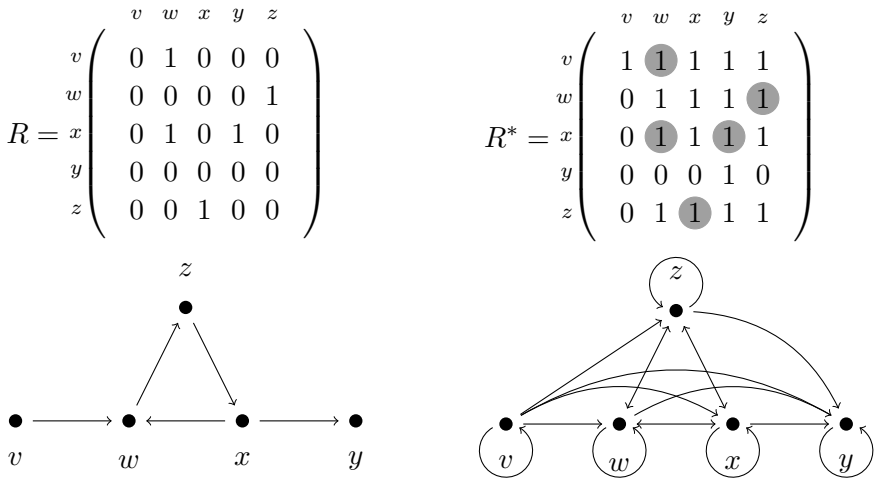


Figure 2.9: An interpretation of the reflexive-transitive closure for digraphs. In the matrix, R^*, the original arcs from the graph of R are highlighted gray.

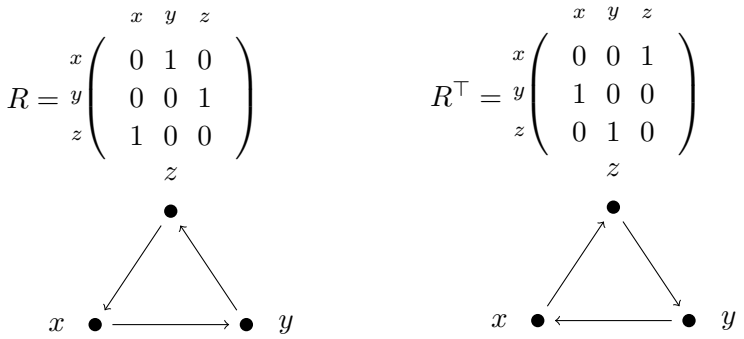


Figure 2.10: An interpretation of relation transposition for digraphs.

It follows that $R^+ = RR^*$. An interpretation of the reflexive-transitive closure, R^*, for digraphs is the reachability relation of the graph of R, that is, $(x, y) \in R^*$ says that vertex y is reachable from x by taking zero or more arcs. For example, in Figure 2.9, vertex x is reachable from vertex v in the graph of R in three steps and there is a transitive arc in R^* from v to x.

The *transpose relation*, $R^\top$, of a relation R is defined as

$$R^\top = \{(y, x) \mid (x, y) \in R\}$$

An interpretation of this for digraphs is to reverse the directions of the arcs of the graph. For example, consider the relation R and its transpose $R^\top$ in Figure 2.10.

A relation is said to be *symmetric* if $R \subseteq R^\top$, *asymmetric* if $R \cap R^\top \subseteq \mathsf{O}$, and *antisymmetric* if $R \cap R^\top \subseteq \mathsf{I}$. The *symmetric closure* of a relation, R, is $R \cup R^\top$ and is the smallest relation that contains R and is symmetric.

Of particular interest are *equivalence relations*. An equivalence relation on a set is a binary relation that is reflexive, symmetric and transitive. Such relations may be used to describe how some members of a set have a common property. This could be used to represent the components of a graph, partitioning connected vertices into classes. For example, Figure 2.11 shows a digraph of a relation R, and its symmetric-reflexive-transitive closure, $(R \cup R^\top)^*$. This describes which vertices are reachable by taking any number of steps, ignoring direction.

A relation, R, is called a vector if $R = R\mathsf{L}$, and a *co-vector* if $R = \mathsf{L}R$. A vector is row constant, that is, every column is identical. A vector, $R \subseteq S \times S$ can be used to represent the subset of S containing those elements $x \in S$ such that $(x, y) \in S$ for all y. This is useful for us to denote a subset of the vertices of a graph.

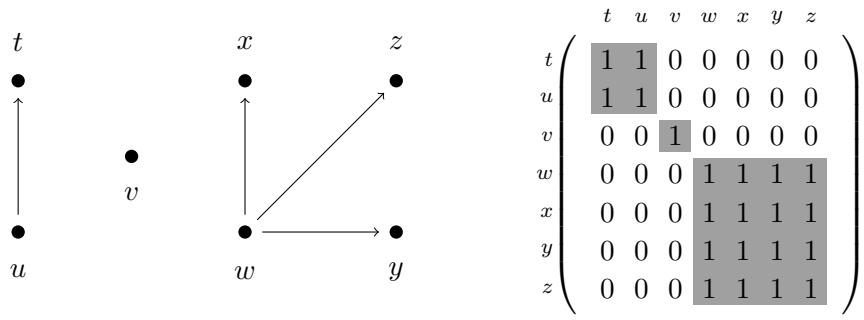


Figure 2.11: How equivalence relations may be used to represent the components of a digraph. The adjacency matrix on the right is the equivalence relation of the graph on the left.

A relation, R, is *univalent* if $R^\top R \subseteq \mathsf{I}$, *injective* if $RR^\top \subseteq \mathsf{I}$, *total* if $\mathsf{I} \subseteq RR^\top$, *surjective* if $\mathsf{I} \subseteq R^\top R$, a *mapping* if R is total and univalent, *bijective* if R is injective and surjective, and a *point* if R is a bijective vector.

These properties have useful interpretations for digraphs. For example, every vertex of the graph of a univalent relation is the source of at most one arc. A total relation may be interpreted as a graph where all vertices are the source of at least one arc. Every vertex of the graph of a bijective mapping is the source and target of exactly one arc, and a point represents a subset of a single element, that is, one vertex of the graph.

Notably, an injective relation may be interpreted as a graph where all vertices are the target of at most one arc. This forms part of the definition of an arborescence, and as we will see in Section 2.3.6, a rooted directed forest.

Binary relations are not so suitable for reasoning about weighted graphs. If a matrix of weights were used to represent a weighted graph then it is not clear what interpretation should be had for the complement operation. For this reason, alternative algebraic structures have been proposed to reason about weighted graphs. These structures share many properties of relations. We present definitions for these structures beginning with orders.

2.3.2 Orders

Several textbooks further discuss the properties of orders [8, 11, 21, 47, 76].

We use order in the sense of the arrangement of things in relation to each other, rather than the sense of a command or instruction. An order may be assigned to mathematical objects. Consider some $a, b \in \mathbb{N}$. An order may be applied to a and b with the usual sense of 'less than': either $a \leq b$ or $b < a$, which is to be read as a is less than or equal to b or b is strictly less than a. Similar to how we might order objects in the world, the 'less than' operation is not the only useful operation by which to order mathematical objects.

If we take the order of sets to be subset inclusion, we can have difficulty making comparisons between some finite $S, Q \subseteq \mathbb{N}$. For example, $S = \{1, 2\}$ and $Q = \{3\}$ are not $\subseteq$-comparable. However, if we were to use the cardinality of the sets as our metric then we can compare S and Q.

Therefore, an order requires some operation by which to compare objects. Also, notice that it does not make sense to assign an order to a single object, rather an order appears as the result of a binary operation: a comparison between two objects.

Definition 1. *An order (also partial order) is a structure $\langle S\,,\ \leq\rangle$, where S is a set and $\leq$ is a binary relation on S that is reflexive, antisymmetric, and transitive. That is, for all $x, y, z \in S$*

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

This is similar to the properties of the equivalence relation, with the antisymmetry property replacing the symmetry property.

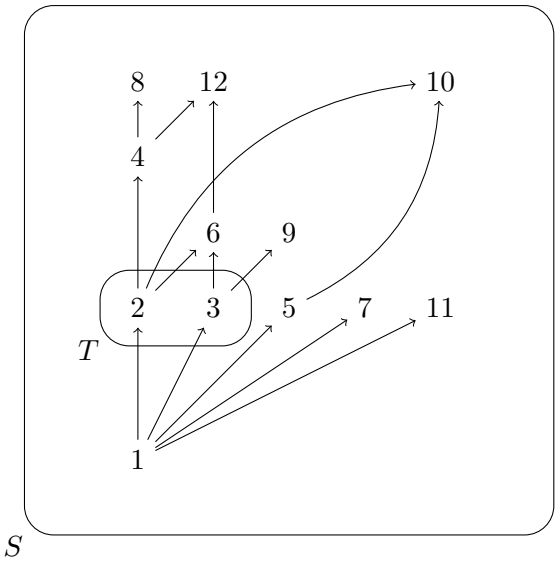


Figure 2.12: A set, S, with a partial order of divisibility, and a subset, T. An arc between two numbers denotes that the source divides the target. The transitive and reflexive arcs are not shown.

Let $P = \langle S\,,\ \leq\rangle$ be a partial order and have $T \subseteq S$. We call an element $x \in S$ an *upper bound* of T if for all $y \in T$, $y \leq x$. Consider such an upper bound, j. We call j the *join* of T if for all $u \in S$, where u is an upper bound of T, $j \leq u$.

Dually, we call an element $x \in S$ a *lower bound* of T if for all $y \in T$, $x \leq y$. A lower bound, m, is called the *meet* of T if for all $l \in S$, where l is a lower bound of T, $l \leq m$. The meet and join are unique if they exist.

For example, let P be a partial order on the set, S, of the positive integers from one to twelve, ordered by divisibility, that is, $a|b \Leftrightarrow (b = ak$ for some integer $k)$. Furthermore, let $T = \{2, 3\} \subseteq S$ as depicted in Figure 2.12. Then, there are two upper bounds of T with respect to the order P, 6 and 12. The join is 6. There is only one lower bound of T with respect to the order P, that is 1 which is also the meet.

2.3.3 Lattices

Lattices have been well studied and many textbooks discuss their structure [2, 8, 11, 21, 35, 36, 76].

If all pairs of elements in a partial order have a meet (also greatest lower bound or infimum) and a join (also least upper bound or supremum) then we call this a lattice.

Definition 2. *A lattice, $\langle S\,,\ \sqcup\,,\ \sqcap\rangle$, is a partial order, $\langle S\,,\ \leq\rangle$, where for all $x, y \in S$, both a meet, $x \sqcap y$, and a join, $x \sqcup y$, exist.*

If L is a lattice, then the following properties hold for all $x, y, z \in L$. Both meet and join are *associative* $x \sqcup (y \sqcup z) = (x \sqcup y) \sqcup z$ and $x \sqcap (y \sqcap z) = (x \sqcap y) \sqcap z$; *commutative*, $x \sqcup y = y \sqcup x$ and $x \sqcap y = y \sqcap x$; and *idempotent* $x \sqcup x = x$ and $x \sqcap x = x$. In addition, $x \leq y$ if and only if $x \sqcap y = x$, and $x \leq y$ if and only if $x \sqcup y = y$; and the *absorption* laws are satisfied, that is, $x \sqcup (x \sqcap y) = x$ and $x \sqcap (x \sqcup y) = x$.

For example, the partially ordered set, $\langle \mathbb{N}\,,\ \leq\rangle$, is a lattice. This lattice has $x \sqcup y = \max(x, y)$ and $x \sqcap y = \min(x, y)$ for all $x, y \in \mathbb{N}$. It is quite easy to show examples of properties holding for this lattice, for example, associativity of meet: $\min(1, \min(2, 3)) = \min(\min(1, 2), 3) = 1$.

Given a partial order $\langle S\,,\ \leq\rangle$, the $\leq$-*greatest* (also top or maximum) element of a subset, P, of S is the $x \in P$ where $y \leq x$ for all $y \in P$. Dually, the $\leq$-*least* (also bottom or minimum)

element is the $x \in P$ where $x \leq y$ for all $y \in P$. Where the order is understood from context then we refer simply to the greatest and least elements.

The greatest and least elements do not necessarily exist. We cannot find the greatest element in $\mathbb{N}$, with respect to the order $\langle \mathbb{N}, \leq \rangle$, though its least element is 0. Since a lattice is an order, we can talk about the least and greatest elements in lattices. Where such elements do exist for the entire lattice, it is called a *bounded lattice*.

Definition 3. *A bounded lattice, $\langle S, \sqcup, \sqcap, \bot, \top \rangle$, is a lattice, $\langle S, \sqcup, \sqcap \rangle$, with a least element, $\bot$, and a greatest element, $\top$.*

For example, the power set, $\mathcal{P}(S)$, ordered by the subset relation, $\langle \mathcal{P}(S), \subseteq \rangle$, forms a structure $\langle \mathcal{P}(S), \cup, \cap, \emptyset, S \rangle$ that is an instance of a bounded lattice. The full set S is the $\subseteq$-greatest element because $S \in \mathcal{P}(S)$ and for all $T \in \mathcal{P}(S)$, $T \subseteq S$. A similar argument describes how $\emptyset$ is the $\subseteq$-least element.

We consider additional axioms that can be asserted for the meet and join operations to give the definition for a *bounded distributive lattice*.

Definition 4. *A bounded distributive lattice, $\langle S, \sqcup, \sqcap, \bot, \top \rangle$, is a bounded lattice where for all $x, y, z \in S$*

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

2.3.4 Relation algebras

In Section 2.3.1 we mentioned that relations have been used to reason about unweighted graphs with some success but that weighted graphs presented a problem as it is unclear how the complement operation should be applied to elements of the set that the relation is formed over. Here we give more precise definitions of the algebras relevant for unweighted graphs to be compared with the structures that we will be working in.

Definition 5. *A* Boolean algebra*, $\langle S, \sqcup, \sqcap, \overline{\ }, \bot, \top \rangle$, is a bounded distributive lattice, $\langle S, \sqcup, \sqcap, \bot, \top \rangle$, with a complement operation, $\overline{\ }$, where for all $x \in S$*

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

There are alternative axiomatizations of Boolean algebras, for example in [2, 21, 50, 59]. The axioms we give, from [76], are presented in terms of bounded distributive lattices to mirror the axiomatization of pseudo-complemented algebras presented in section 2.3.5.

Definition 6. *A* relation algebra*, $\langle S, \sqcup, \sqcap, \cdot, \overline{\ }, {}^{\top}, \bot, \top, 1 \rangle$, is a Boolean algebra, $\langle S, \sqcup, \sqcap, \overline{\ }, \bot, \top \rangle$, with composition, $\cdot$, and transpose, ${}^{\top}$, operations, and a constant 1, where for all $x, y, z \in S$*

$$x(yz) = (xy)z \tag{2.1}$$
$$(x \sqcup y)z = xz \sqcup yz \tag{2.2}$$
$$z(x \sqcup y) = zx \sqcup zy \tag{2.3}$$
$$x1 = x \tag{2.4}$$
$$1x = x \tag{2.5}$$
$$x^{\top\top} = x \tag{2.6}$$
$$(x \sqcup y)^{\top} = x^{\top} \sqcup y^{\top} \tag{2.7}$$
$$(xy)^{\top} = y^{\top}x^{\top} \tag{2.8}$$
$$x^{\top}\overline{xy} \leq \overline{y} \tag{2.9}$$

We note that, in this thesis, composition, $x \cdot y$, is often abbreviated to xy. We also use this abbreviation for other algebras that have a composition operation.

The axiomatization in Definition 6 is due to Jónsson and Tarski as discussed in [57]. Other forms can be found in [58, 81]. Unless overridden with brackets, the operations have the precedence, from highest to lowest: $^\top$, $^-$, $\cdot$, $\sqcap$, $\sqcup$.

A number of properties hold in relation algebras and we give two examples here. If R is a relation algebra then the following properties hold for all $x, y, z \in R$. Composition subdistributes over meet, $x(y \sqcap z) \leq xy \sqcap xz$ and $(y \sqcap z)x \leq yx \sqcap zx$; and join is $\leq$-*isotone*, $x \leq y$ implies $z \sqcup x \leq z \sqcup y$. Further discussion of the properties of relation algebras can be found in, for example, [77].

2.3.5 Stone relation algebras

For weighted graphs, the idea proposed by Guttmann [38] is to work in relation algebras where the Boolean algebra reduct is weakened only so much as to permit the inclusion of arc weights in the considered set while maintaining most of the structure of a relation algebra. Stone relation algebras fit this purpose.

Some of the following algebraic structures are discussed in a number of textbooks [2, 11, 35, 36]. Stone algebras are additionally treated in [17, 37, 42]. Stone relation algebras are introduced in [43] and further discussed in [42, 44].

Definition 7. *A* distributive p-algebra, $\langle S, \sqcup, \sqcap, ^-, \bot, \top\rangle$, *is a bounded distributive lattice,* $\langle S, \sqcup, \sqcap, \bot, \top\rangle$, *with a pseudo-complement operation,* $^-$, *where for all* $x, y \in S$

$$x \sqcap y = \bot \Leftrightarrow x \leq \overline{y}$$

This says $\overline{y}$ is the $\leq$-*greatest* element whose meet with y is $\bot$.

Definition 8. *A* Stone algebra *is a distributive p-algebra,* $\langle S, \sqcup, \sqcap, ^-, \bot, \top\rangle$, *where for all* $x \in S$

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

If $x = \overline{\overline{x}}$ then x is said to be *regular*. If a Stone algebra has only regular elements then it is a Boolean algebra.

A number of properties hold in Stone algebras and we give two examples here. If S is a Stone relation algebra then the following properties hold for all $x, y \in S$. Complement is $\leq$-*antitone*, $x \leq y$ if and only if $\overline{y} \leq \overline{x}$; and $\overline{\overline{\overline{x}}} = \overline{x}$. Further discussion of the properties of Stone algebras can be found in the resources listed at the start of this section.

A Stone algebra is a generalization of a Boolean algebra with a weakened complement operation. Therefore, a trivial example of a Stone algebra is a Boolean algebra with the complement operation playing the role of the pseudo-complement operation. Often, the MST problem is solved for graphs with arc weights selected over $\mathbb{R}$. Therefore, it is sensible for us to consider an example that uses this set, such as Theorem 5 from [42].

Consider the real numbers extended by $\bot$ and $\top$: $\mathbb{R}' = \mathbb{R} \cup \{\bot, \top\}$. Here, $\bot$ denotes the non-existence of an arc, $\top$ denotes an arc of unknown weight, and the real numbers denote weights of their corresponding values. If we define the ordered set, $\langle \mathbb{R}', \leq\rangle$, such that $\bot$ is the $\leq$-least element and $\top$ is the $\leq$-greatest element then $\langle \mathbb{R}', \max, \min, ^-, \bot, \top\rangle$ is a Stone algebra. The effect of Definition 7 on the real numbers is that, for all $x \in \mathbb{R}'$

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

While some examples are given in [2], we are motivated by being able to give precise mathematical descriptions of graphs and operations on graphs. For this purpose, we give an example of this algebra for matrices whose entries range over the extended real numbers, $\mathbb{R}'$.

We denote the set of all square matrices whose entries range over the extended real numbers as $\mathbb{R}'^{A\times A}$, where A is the set of indices of the matrix. An intuition for this notation is given in Appendix A. We interpret any $M \in \mathbb{R}'^{A\times A}$ as a graph with arc weights taken from $\mathbb{R}'$ and with vertex set A. Under this interpretation, $\langle \mathbb{R}'^{A\times A}, \sqcup, \sqcap, \overline{\ }, \bot, \top\rangle$ is a Stone algebra where the operations, $\sqcup$, $\sqcap$, $\overline{\ }$, $\bot$, $\top$, and the lattice order, $\leq$, are lifted componentwise so that

$$\begin{aligned}(M \sqcup N)_{i,j} &= M_{i,j} \sqcup N_{i,j}\\ (M \sqcap N)_{i,j} &= M_{i,j} \sqcap N_{i,j}\\ \overline{M}_{i,j} &= \overline{M_{i,j}}\\ \bot_{i,j} &= \bot\\ \top_{i,j} &= \top\end{aligned}$$

and $M \leq N \Leftrightarrow \forall i,j \in A\colon M_{i,j} \leq N_{i,j}$. In this instance of a Stone algebra, the regular elements are matrices with entries that are either $\bot$ or $\top$. These elements can be interpreted as matrices, without weight information, that describe a graph's structure. The structure of a graph, M, can be obtained by applying the complement twice: $\overline{\overline{M}}$.

Definition 9. *A Stone relation algebra, $\langle S, \sqcup, \sqcap, \cdot, \overline{\ }, {}^\top, \bot, \top, 1\rangle$, is a Stone algebra with composition and transpose operations denoted in the same manner as for relations and a constant, 1, where for all $x, y, z \in S$*

$$\begin{aligned}(xy)z &= x(yz)\\ 1x &= x\\ (x \sqcup y)z &= xz \sqcup yz\\ (xy)^\top &= y^\top x^\top\\ (x \sqcup y)^\top &= x^\top \sqcup y^\top\\ x^{\top\top} &= x\\ \bot x &= \bot\\ xy \sqcap z &\leq x(y \sqcap x^\top z)\\ \overline{\overline{xy}} &= \overline{\overline{x}}\,\overline{\overline{y}}\\ \overline{\overline{1}} &= 1\end{aligned}$$

Unless overridden with brackets, the operations have the precedence, from highest to lowest: $^\top$, $\overline{\ }$, $\cdot$, $\sqcap$, $\sqcup$.

Recall, from Section 2.3.4, that a relation algebra has the signature, $\langle S, \sqcup, \sqcap, \cdot, \overline{\ }, {}^\top, \bot, \top, 1\rangle$. A Stone relation algebra has the same signature and many properties of relation algebras hold for Stone relation algebras, some of which are given in Theorems 8 and 9 of [42]. Indeed, if the Boolean algebra reduct of a relation algebra, $\langle S, \sqcup, \sqcap, \overline{\ }, \bot, \top\rangle$, is replaced with a Stone algebra then we obtain a Stone relation algebra. We take the opportunity to reuse relevant terminology.

An element $x \in S$ is called *reflexive* if $1 \leq x$, *transitive* if $xx \leq x$, *symmetric* if $x = x^\top$, a *vector* if $x\top = x$, a *co-vector* if $\top x = x$, *co-reflexive* if $x \leq 1$, *irreflexive* if $x \leq \overline{1}$, *asymmetric* if $x \sqcap x^\top = \bot$, *antisymmetric* if $x \sqcap x^\top \leq 1$, *univalent* if $x^\top x \leq 1$, *injective* if $xx^\top \leq 1$, *total* if $1 \leq xx^\top$, *surjective* if $1 \leq x^\top x$, a *mapping* if x is univalent and total, *bijective* if x is injective and surjective, a *point* if x is a bijective vector, and an *arc* if both $x\top$ and $x^\top\top$ are bijective.

For example, consider the Stone algebra, $\mathbb{R}'^{A\times A}$, previously discussed, extended with the operations of a Stone relation algebra. Then $\langle \mathbb{R}'^{A\times A}, \sqcup, \sqcap, \cdot, \overline{\ }, {}^{\top}, \bot, \top, 1\rangle$ is a Stone relation algebra with

$$\begin{aligned}(MN)_{i,j} &= \max_{k\in A}(\min(M_{i,k}, N_{k,j}))\\ (M^{\top})_{i,j} &= M_{j,i}\\ 1_{i,j} &= \begin{cases}\top & \text{if } i = j\\ \bot & \text{otherwise}\end{cases}\end{aligned}$$

Let M be an element in the Stone algebra, $\mathbb{R}'^{A\times A}$. Then, M is a matrix that represents a weighted graph. Because the entries of M are not selected over a binary set, some of the graph interpretations to be had about Stone algebras are not the same as for relations. Theorem 13 of [44] describes such an M as being

1. reflexive if and only if the diagonal entries are all $\top$,
2. co-reflexive if and only if only the diagonal has non-$\bot$ entries,
3. irreflexive if and only if the diagonal entries are all $\bot$,
4. symmetric if and only if $\forall i, j \in A\colon M_{i,j} = M_{j,i}$.

A symmetric M may be interpreted as an undirected weighted graph. Theorem 11 of [44] describes such an M as being

1. univalent if and only if in every row at most one entry is not $\bot$,
2. injective if and only if in every column at most one entry is not $\bot$,
3. total if and only if in every row at least one entry is $\top$,
4. surjective if and only if in every column at least one entry is $\top$.

The same interpretations are taken for vectors, co-vectors, points, and arcs as for relations. Theorem 12 of [44] describes M as being

1. a vector, if and only if in every row no entries are different,
2. a co-vector, if and only if in every column no entries are different,
3. a point, if and only if exactly one row has all entries $\top$ and every other row has only $\bot$ entries, and
4. an arc, if and only if exactly one entry is $\top$ and all others are $\bot$.

Therefore, points and arcs are regular but vectors and co-vectors merely require constant rows and columns respectively.

For some properties of relations that do not hold in Stone relation algebras, there exist similar, weakened properties that do hold. For example, the Schröder equivalence $x^{\top}y \leq z \Leftrightarrow x\overline{z} \leq \overline{y}$ holds in relation algebras but not in Stone relation algebras. However, if the right-hand sides of the inequalities are regular then the equivalence holds. This can be achieved with the pseudo-complement, $x^{\top}y \leq \overline{z} \Leftrightarrow xz \leq \overline{y}$.

2.3.6 Stone-Kleene relation algebras

Stone relation algebras are extended to Stone-Kleene relation algebras in [43]. This allows us to reason about reachability in graphs, conceptually similar to how it was discussed in Section 2.3.1. The unfold and induction axioms given for the Kleene star that are used in Definition 10 are taken from [53].

Definition 10. *A* Stone-Kleene relation algebra, $\langle S, \sqcup, \sqcap, \cdot, \overline{\ }, {}^{\top}, {}^*, \bot, \top, 1\rangle$, *is a Stone relation algebra,* $\langle S, \sqcup, \sqcap, \cdot, \overline{\ }, {}^{\top}, \bot, \top, 1\rangle$, *with an operation,* *, *where for all* $x, y, z \in S$ *the unfold and induction axioms hold*

$$1 \sqcup xx^* \leq x^* \qquad z \sqcup yx \leq x \Rightarrow y^*z \leq x$$
$$1 \sqcup x^*x \leq x^* \qquad z \sqcup xy \leq x \Rightarrow zy^* \leq x$$

and additionally,

$$(\overline{\overline{x}})^* = \overline{\overline{x^*}} \tag{2.10}$$

We abbreviate xx^* as x^+. Furthermore, we call any $x \in S$ *acyclic* if $x^+ \leq \overline{1}$, and a *rooted directed forest* if x is injective and acyclic. Axiom (2.10) states that the regular elements of S are closed under the * operation.

Consider the Stone relation algebra, $\langle \mathbb{R}'^{A\times A}, \sqcup, \sqcap, \cdot, \overline{\ }, {}^{\top}, \bot, \top, 1\rangle$, extended with the * operation of a Stone-Kleene relation algebra where A is finite, * is defined recursively using Conway's construction [19]

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^* = \begin{pmatrix} (a \sqcup bd^*c)^* & a^*b(d \sqcup ca^*b)^* \\ d^*c(a \sqcup bd^*c)^* & (d \sqcup ca^*b)^* \end{pmatrix}$$

and for $x \in \mathbb{R}'$, $x^* = \top$.

This forms a Stone-Kleene relation algebra, $\langle \mathbb{R}'^{A\times A}, \sqcup, \sqcap, \cdot, \overline{\ }, {}^{\top}, {}^*, \bot, \top, 1\rangle$. We extend the weighted graph interpretation, from Section 2.3.5, for an element, $M \in \mathbb{R}'^{A\times A}$ over such a Stone-Kleene relation algebra. If M is a rooted directed forest then it represents a graph that is acyclic and whose edges form directed paths that are directed away from their root vertices. As described in Section 2.1.2, an alternative description for a rooted directed forest would be to have the directed paths directed towards their root vertices. This would be acyclic and univalent.

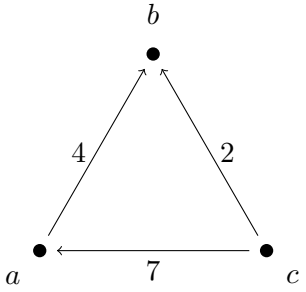
Borůvka's MST algorithm selects edges of minimum weight so that we need to be able to reason about the comparison of edge weights to complete our proof. Algebras for summing and minimizing weights are given in Definition 17 of [44]; in particular, we use the s and m operations of m-Kleene algebras.

Definition 11. *An* m-Kleene algebra, $\langle S, \sqcup, \sqcap, \cdot, +, \overline{\ }, {}^{\top}, {}^*, s, m, \bot, \top, 1\rangle$, *is a Stone-Kleene relation algebra,* $\langle S, \sqcup, \sqcap, \cdot, \overline{\ }, {}^{\top}, {}^*, \bot, \top, 1\rangle$, *with addition,* $+$, *summation,* s, *and minimum selection,* m, *operations, where for all* $x, y, z \in S$, *the summation properties are satisfied*

$$x \neq \bot \wedge s(x) \leq s(y) \Rightarrow s(z) + s(x) \leq s(z) + s(y) \tag{2.11}$$
$$s(x) + s(\bot) = s(x) \tag{2.12}$$
$$s(x) + s(y) = s(x \sqcup y) + s(x \sqcap y) \tag{2.13}$$
$$s(x^{\top}) = s(x) \tag{2.14}$$

the linear properties are satisfied

$$s(x) \leq s(y) \vee s(y) \leq s(x) \tag{2.15}$$
$$\{\overline{x} \mid x \in S\} \text{ is finite} \tag{2.16}$$



(a) A digraph, G, that has three edges with weights: 2, 4, and 7.

$$G = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} \bot & 4 & \bot \\ \bot & \bot & \bot \\ 7 & 2 & \bot \end{pmatrix} \end{array}$$

$$m(G) = \begin{array}{c} \\ a \\ b \\ c \end{array} \begin{array}{c} \begin{array}{ccc} a & b & c \end{array} \\ \begin{pmatrix} \bot & \bot & \bot \\ \bot & \bot & \bot \\ \bot & \top & \bot \end{pmatrix} \end{array}$$

(b) The graph from Figure 2.13(a) in matrix form and the output of the m operation on G.

Figure 2.13: An example of the m operation on a graph.

and the minimum selection properties are satisfied

$$m(x) \leq \overline{\overline{x}} \tag{2.17}$$

$$x \neq \bot \Rightarrow m(x) \text{ is an arc} \tag{2.18}$$

$$y \text{ is an arc} \wedge y \sqcap x \neq \bot \Rightarrow s(m(x) \sqcap x) \leq s(y \sqcap x) \tag{2.19}$$

We discuss the axioms that are more important to our proof. Further details about the remaining axioms are provided in [44].

Axiom (2.14) ensures that the weights of arcs do not depend on direction. Axiom (2.17) states that, ignoring weight, the output of the m operation is contained in the graph. Axiom (2.18) states that if a graph is not empty then the result of the m operation is an arc. If there is more than one minimum-weight arc in the graph, the order of row and column indices can be used to uniquely identify an arc. Axiom (2.19) ensures that the weight of arc $m(x)$ must be less than or equal to the weight of any other arc in the graph.

For the $\mathbb{R}'^{A\times A}$ model, the output of the m operation on a matrix whose entries are not all $\bot$ is a matrix with an entry of $\top$ in the row and column denoting an arc with minimum weight. For example, Figure 2.13 shows the output of the m operation for a weighted graph. In Figure 2.13(b), the arc with weight 2 has been selected for the graph shown in Figure 2.13(a). There is only one entry in $m(G)$, that is $\top$, in row c and column b, all other entries are $\bot$. The axioms of the m operation are satisfied: $m(G)$ is contained in $\overline{\overline{G}}$, is an arc, and is the minimum weight arc from G.

Chapter 3

Formalization of Borůvka's MST algorithm

In this chapter, we give our formalization of Borůvka's MST algorithm.

In Section 3.1 we introduce an operation, k, that axiomatizes component selection in a graph with k-Stone relation algebras. This operation is used in our formalization. Additionally, we define m-k-Stone-Kleene relation algebras, which our proof is completed in, and which combine k-Stone relation algebras with m-Kleene algebras and include the Tarski rule. We give a simple example of proving a result in m-k-Stone-Kleene relation algebras at the end of Section 3.3.

To formalize the algorithm some work was done to massage the method described in Section 2.2.2 into a form that is more amenable to being described with the language available to us. This is discussed further in Section 3.2, where we also present the formalization. The formalization is discussed in Section 3.3.

3.1 An operation to select components

Recall, from Section 2.2.2, that Borůvka's MST algorithm iterates over the trees of a forest. We treat these trees as components of the forest, so to achieve this behavior, we extend Stone relation algebras with a binary operation, k, that models component selection.

This operation was added so that we could create a formalization of Borůvka's MST algorithm that is similar to its textual description. Our first approach to formalize component selection used the m operation described in Section 2.3.6

$$c = h^{\top *} h^* m(j)\top$$

Here, h is a forest representing the spanning tree under construction and j is a set of vertices that have yet to be processed by the algorithm. Therefore, the expression, c, is a set of vertices, represented as a vector, that are connected in h. Additionally, the set contains a vertex that still needs to be processed, selected by $m(j)\top$. This approach to select a component is subpar for two reasons. Firstly, it is not immediately clear that this expression selects a component of the graph. In contrast, the k operation promises to return an arbitrary component of a graph, so its meaning is clear. Secondly, the properties that are required of selected components in the proof need to be derived from the algebraic expression. This requires additional reasoning, while the k operation ensures those properties by its axioms.

Given a set of vertices and some information about their connections, the k operation outputs an arbitrary component of those vertices represented as a vector.

Definition 12. *A* k-Stone relation algebra, $\langle S, \sqcup, \sqcap, \cdot, \bar{}, {}^{\top}, k, \bot, \top, 1\rangle$, *is a Stone*

relation algebra, $\langle S\,,\ \sqcup\,,\ \sqcap\,,\ \cdot\,,\ ^{-}\,,\ ^{\top}\,,\ \bot\,,\ \top\,,\ 1\rangle$*, with an operation* k*, where for all* $x, y \in S$

$$k(x,y) = k(x,y)\top \tag{3.1}$$
$$k(x,y) \leq \overline{\overline{y}} \tag{3.2}$$
$$k(x,y) = \overline{\overline{k(x,y)}} \tag{3.3}$$
$$k(x,y) \cdot k(x,y)^{\top} \leq x \tag{3.4}$$
$$k(x,y) = x \cdot k(x,y) \tag{3.5}$$

and, if x *is a regular equivalence,* y *is a regular vector,* $xy = y$*, and* $y \neq \bot$ *then*

$$k(x,y) \neq \bot \tag{3.6}$$

If x is a regular equivalence and y is a regular vector then the $k(x,y)$ operation may be used to choose an arbitrary component. In this case, y is the set of vertices from which to select a component and x describes connectivity among those vertices.

Axiom (3.1) states that the image of the k operation is a vector.

Axiom (3.2) expresses that the result of k is contained in the set of vertices we are selecting from, ignoring the weights. For example, we see that e is contained in y in Figures 3.1(c) and 3.1(d).

Axiom (3.3) ensures that the output of k is regular.

Axiom (3.4) makes any two vertices from the result of k connected in x. For example, consider Figure 3.1(e), the graphical representation of the vector, $k(x,y)$ from Figure 3.1(d). For any two steps that we can take first forwards, $k(x,y)$, and then backwards, $k(x,y)^{\top}$, we will start and arrive in vertices e or f, which are in the same component of x.

Axiom (3.5) expresses that the result of k is closed under being connected in x. This means that either all vertices of a component of x are included in the output of k, or none are.

Axiom (3.6) requires that k returns a non-empty component if the input satisfies the given criteria. The criterion that $xy = y$ ensures that y contains each component entirely. If any of the criteria are not met the k operation may return $\bot$.

The following result shows that one instance of k-Stone relation algebras may be obtained from m-Kleene algebras.

Theorem 1. *Let* S *be an* m*-Kleene algebra with a* k *operation defined as*

$$k(x,y) = \begin{cases} x \cdot m(y)\top & \text{if } x \text{ is a regular equivalence and } y \neq \bot \\ & \quad \text{and } y \text{ is a regular vector and } xy = y, \\ \bot & \text{otherwise} \end{cases} \tag{3.7}$$

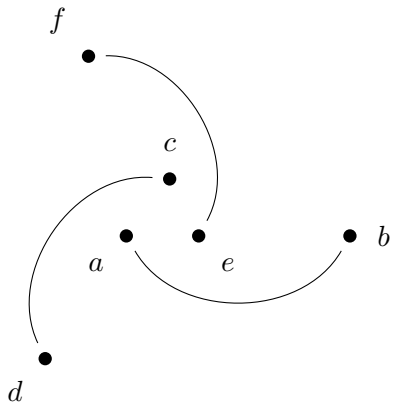
then S *is a* k*-Stone relation algebra.*

Finally, we combine k-Stone relation algebras with m-Kleene algebras and include the Tarski rule for regular elements.

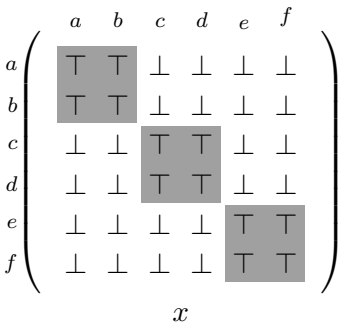
Definition 13. *An* m*-*k-Stone-Kleene relation algebra*,* $\langle S\,,\ \sqcup\,,\ \sqcap\,,\ \cdot\,,\ +\,,\ ^{-}\,,\ ^{\top}\,,\ ^{*}\,,\ s\,,\ m\,,\ k\,,\ \bot\,,\ \top\,,\ 1\rangle$*, is an* m*-Kleene algebra,* $\langle S\,,\ \sqcup\,,\ \sqcap\,,\ \cdot\,,\ +\,,\ ^{-}\,,\ ^{\top}\,,\ ^{*}\,,\ s\,,\ m\,,\ \bot\,,\ \top\,,\ 1\rangle$*, with a component selection operation,* k*, such that the reduct* $\langle S\,,\ \sqcup\,,\ \sqcap\,,\ \cdot\,,\ ^{-}\,,\ ^{\top}\,,\ k\,,\ \bot\,,\ \top\,,\ 1\rangle$ *is a* k*-Stone relation algebra and for all* $x \in S$ *the Tarski rule,*

$$\overline{\overline{x}} = x \wedge x \neq \bot \Rightarrow \top x \top = \top \tag{3.8}$$

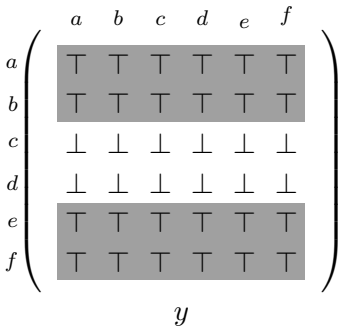
is satisfied.



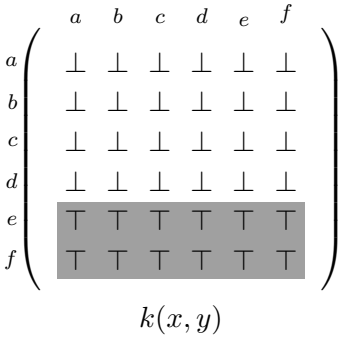
(a) A graph with three components.



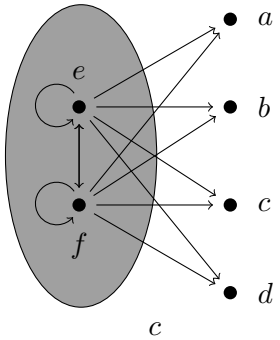
(b) The equivalence, x, represents the components of the graph.



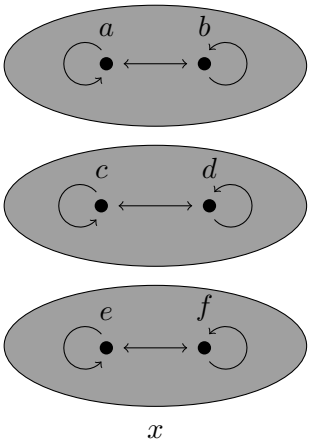
(c) The vector, y, represents the set of vertices that a particular component will be selected from.



(d) The k operation selects a component from the vertices of y, that are connected in x.



(e) A graphical view of the selected component, $k(x, y)$.



(f) A graphical view of the equivalence, x.

Figure 3.1: The component selection operation, k, operating on a set of nodes, y, that are connected as shown in the top-left graph. The connection information is encoded in the equivalence, x, shown both in matrix and graph form.

3.2 Formalization description

Because we are working in an algebra of matrices over extended real numbers, $\mathbb{R}'^{A\times A}$, our formalization operates on digraphs. However, recall from Section 2.2.2 that Borůvka's MST algorithm operates on an undirected graph. Therefore, to resolve this discrepancy we use symmetric elements of the algebra to represent undirected graphs. In particular, the input digraph, g, is symmetric.

Recall that the arcs of the MST that will be output are tracked by coloring. Some work would be required to associate a coloring property with each arc and vertex by embedding it in the algebra. Instead, our formalization of Borůvka's MST algorithm maintains a rooted directed forest variable, f, to be output upon termination. In our implementation, the variable f may be thought of as those colored arcs, though we will not refer to coloring further. A forest (undirected) may be obtained from f by taking the symmetric closure, $f \sqcup f^\top$.

Additionally, if g is connected then Borůvka's MST algorithm outputs a MST. However, if g is not connected then the output is a minimum spanning forest. By requiring that the output be a minimum spanning forest we obtain a proof for a more general specification.

We do not require that the input graph's arc weights are distinct. Instead, our formalization performs a check to ensure that a cycle will not be created when adding an arc to the forest.

With these considerations in mind, we re-describe the algorithm.

Borůvka's MST algorithm takes as input an undirected graph, g. Initialize a rooted directed forest, f, where each vertex in g is a tree with no arcs. Repeat the following step while there are still arcs, in g, between components, in f.

For every component, c, in f,

1. Determine those arcs incident to vertices in c that do not have a source and target in c.
2. Select from those arcs one with minimum-weight and add it to f.

Adding an arc to f in step 2 does not alter the components being iterated over in the inner loop. The algorithm outputs f.

This algorithm is formalized in Figure 3.2. We begin with a high-level description of its operation and continue with a more detailed discussion in the following sections.

The input to the algorithm is a symmetric, weighted graph, g, (line 1). The variable f is initialized as empty (line 2). It will become a structural representation of the minimum spanning forest of the graph so it will be regular.

The outer while-loop continues until there are no arcs between the components of the rooted directed forest (line 3). Lines 4 and 5 initialize variables that are used by the inner while-loop. The regular vector, j, tracks the set of vertices that have yet to be considered by the inner while-loop. The rooted directed forest, h, is used to maintain a stable representation of what f was for each iteration of the outer loop.

On line 6, the variable d is initialized. This variable tracks the arcs that have been added to f in each iteration of the outer loop (line 13). While d is not required by the algorithm it is used in the correctness proof.

The inner while-loop terminates when all components have been processed (line 7). An arbitrary component, c, is chosen among those that have not been processed (line 8). We select a minimum-weighted arc, e, whose source is inside c and whose target is outside c (line 9).

Recall, from Section 2.2.2, that Borůvka's MST algorithm requires the input graph's arc weights to be distinct. Because our formalization does not require this, we have added a check in the inner loop to ensure that a cycle is not created. We check that e is not contained in a component of f (line 10) and make no adjustment to f in the current iteration of the inner loop if it is (line 15). In future work, this check should be able to be removed and we discuss this limitation in Section 5.1.

```
1  input g
2  f ← ⊥
3  while \overline{f^{⊤*} f^*} ⊓ g ≠ ⊥ do
4  |    j ← ⊤
5  |    h ← f
6  |    d ← ⊥
7  |    while j ≠ ⊥ do
8  |    |    c ← k(h^{⊤*} h^*, j)
9  |    |    e ← m(c c̄^⊤ ⊓ g)
10 |    |    if e ≤ \overline{f^{⊤*} f^*} then
11 |    |    |    f ← f ⊓ ē^⊤
12 |    |    |    f ← (f ⊓ \overline{⊤ e f^{⊤*}}) ⊔ (f ⊓ ⊤ e f^{⊤*})^⊤ ⊔ e
13 |    |    |    d ← d ⊔ e
14 |    |    else
15 |    |    |    skip
16 |    |    end
17 |    |    j ← j ⊓ c̄
18 |    end
19 end
20 output f
```

Figure 3.2: A relational formalization of Borůvka's MST algorithm.

Before adding the arc to the rooted directed forest we ensure the arc's transpose is removed, as it may have been added in a previous iteration of the inner while-loop, to mitigate the creation of a cycle (line 11). The minimum-weighted outgoing arc is added to f and at the same time, we reverse any paths that would break the injective property required to maintain that f is acyclic (line 12). We update d to track the arcs that have been added in this iteration of the outer while-loop. We remove the processed component from j so that it is not considered in the next iteration of the inner while-loop (line 17).

When the outer while-loop exits the algorithm terminates returning f (line 20). The forest, f, contains the structural information of the found minimum spanning forest but not the weight information. If desired, this could be obtained by taking the meet with g.

3.3 Operation details

We describe how the major parts of the formalization operate: how the components of the rooted directed forest are represented, how a component is selected from the rooted directed forest in the inner loop, how an arc is selected to be added to f, and the principle behind maintaining the injective property of f. We conclude this section with an example of how results are proved in m-k-Stone-Kleene relation algebras.

Throughout the remainder of this thesis, we often need to refer to variables that have been updated since the previous iteration of the inner loop. This is done with prime notation. For example, $d' = d \sqcup e$, is the value of d at the end of an iteration.

3.3.1 Processing components

The components of f can be represented as an equivalence relation, $(f \sqcup f^\top)^*$, as discussed in Section 2.3.1. Since f is a rooted directed forest and injective, by Theorem 21 of [44], we can instead express the components of f as $f^{\top *} f^*$. Therefore, $\overline{f^{\top *} f^*}$ contains all possible arcs

between components in f and we take the meet with g, in line 3 of the formalization, to consider only those arcs which exist in the graph. As long as such a component exists the outer loop continues.

When we refer to the components of some forest, x, we will abbreviate this using the notation from Definition 20 of [44], that is, $c(x) = x^{\top *} x^*$.

3.3.2 Component selection

In the inner loop we select a component to process from those that still require processing using $c \leftarrow k(h^{\top *} h^*, j)$. The vector j represents the set of vertices not yet processed by the inner loop. The rooted directed forest, h, contains f as it was when the current iteration of the outer loop started. Therefore, $c(h) = h^{\top *} h^*$ is an equivalence relation that describes which vertices were connected when the current iteration of the outer loop started. We record this information using h since Borůvka's MST algorithm needs to process every component of the forest, as it was at the start of the current iteration of the outer loop.

We use the k operation from Section 3.1 to choose an arbitrary component such that it contains only vertices we have not yet processed, in j, and where those vertices were connected when the current iteration of the outer loop started.

On line 17, the vertices in the processed component are removed from j, not to be considered in subsequent iterations of the inner loop. In this way, the inner loop iterates over the components of f as they were before arcs were added between them.

For example, in Figure 3.3 the state of various variables is represented for a graph that has been partially processed by the algorithm in Figure 3.2 where line 8 has just been executed. There are three components in h, depicted by circles in the graph and as a matrix of the equivalence, $c(h)$. One iteration of the inner loop has been completed, where the component, in h, with vertices w and x has been processed. This is evident from the contents of vector j, and the arc (x, y) being in f. A component, $c = k(c(h), j)$, has just been selected and it is the component, in h, with vertices y and z.

3.3.3 Arc selection

The formulation, $m(c\bar{c}^\top \sqcap g)$ was taken from [43], where it is expressed as $m(v\bar{v}^\top \sqcap g)$, and is used in the formalization of Prim's MST algorithm to select an arc with minimum weight that leaves a set of visited vertices, represented by vector v.

In our formulation, c is likewise a vector and $c\bar{c}^\top$ is the set of all possible arcs that have a source in c and a target not in c. Since g is symmetric, we are actually considering the set of all arcs that have one incident vertex inside the component and the other incident vertex outside the component, though our selection is directed.

While the component we select is regular and does not contain weight information, the graph, g, does. So the meet with g not only restricts $c\bar{c}^\top$ to those arcs that are in the graph but also retains the weight information necessary for the m operation to return one of the desired minimum weight arcs, $e = m(c\bar{c}^\top \sqcap g)$ on line 9 of Figure 3.2.

3.3.4 Preservation of injectivity

A particularly important property we maintain is that f is a rooted directed forest. Recall from Section 2.3.6 that a rooted directed forest must be both acyclic and injective. To maintain that f is injective some care must be taken when adding arcs to it. This can be illustrated with an example.

We have reused a formulation given in [44] that maintains the injective property of f when adding an arc e between components of f. In Figure 3.4, a rooted directed forest, f, is depicted before and after an arc, e, is added. The path, p, from a root of the rooted directed forest to

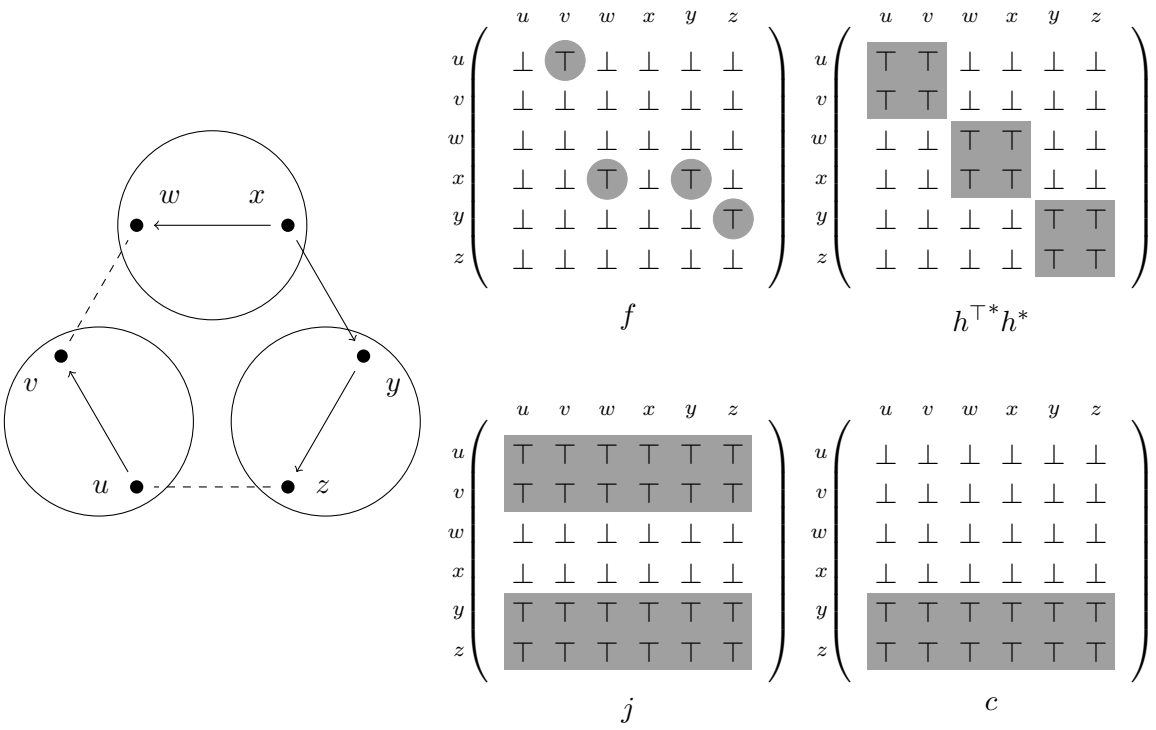


Figure 3.3: An example of component selection for a graph with six vertices and arcs. A solid line represents an arc that has been added to f while a dashed line represents an arc that exists in the graph but has not been added to f. Circles in the graph depict components in h. States of various variables are displayed on the right as matrices. Highlighting is used to help distinguish $\top$ and $\bot$.

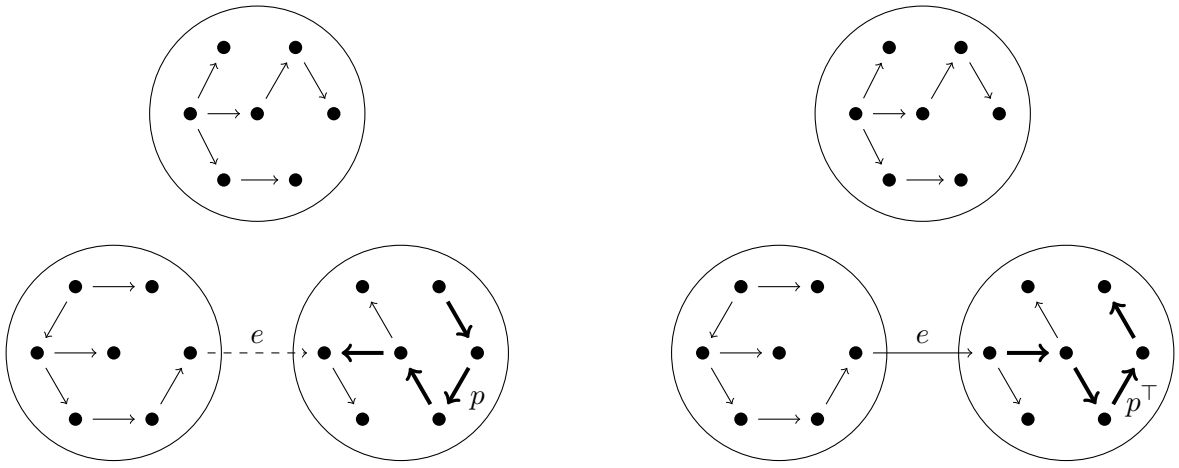


Figure 3.4: The rooted directed forest, f, before and after adding arc, e. The path, p, from the root of the rooted directed forest is reversed to maintain injectivity. A dashed line indicates those arcs in the graph that have not been added to the rooted directed forest. A solid line indicates the arc is in the rooted directed forest. The vertices enclosed in a circle denote a component, in h.

the target of e is

$$p = f \sqcap \top e f^{\top *}$$

This construction can be understood as those arcs in f that are reachable from the target of e by taking zero or more steps backward in f.

When e is added to f, this path must be reversed to maintain the injective property of the rooted directed forest. That is, when adding e, we alter f as follows:

$$\begin{aligned} f' &= (f \sqcap \overline{p}) \sqcup p^{\top} \sqcup e \\ &= (f \sqcap \overline{f \sqcap \top e f^{\top *}}) \sqcup (f \sqcap \top e f^{\top *})^{\top} \sqcup e \end{aligned}$$

This removes the path from the target of e to the root of the rooted directed forest, by taking the meet with $\overline{p}$, and replaces it with the path reversed, by taking the join with $p^{\top}$. This simplifies to the expression in line 12 of Figure 3.2.

Additionally, when adding an arc to f we take the meet with the complement of the arc's transpose to ensure that a cycle is not created (line 11 of the algorithm). This is an artifact of working on a directed graph. As a result, the complete expression that describes how f is modified to f' in the inner loop is

$$f' = \left(f \sqcap \overline{e}^{\top} \sqcap \overline{\top e \left(f \sqcap \overline{e}^{\top}\right)^{\top *}}\right) \sqcup \left(f \sqcap \overline{e}^{\top} \sqcap \top e \left(f \sqcap \overline{e}^{\top}\right)^{\top *}\right)^{\top} \sqcup e$$

3.3.5 Proving properties in m-k-Stone-Kleene relation algebras

We conclude this section by giving an example of how we prove properties about weighted graphs in m-k-Stone Kleene relation algebras. The result presented here is a gentle introduction before encountering the more complex proofs presented in Chapter 4.

Let g be a weighted graph input to the algorithm described in Section 3.2. As the inner loop of that algorithm iterates over the components of the rooted directed forest, an arbitrary component, $c = k(c(h), j)$ is chosen. Then, the arc $e = m(c\overline{c}^{\top} \sqcap g)$, with minimal weight having a source in c and target outside c is selected. The vector j is the set of vertices that have yet to be processed by the inner loop and is not $\bot$. All variables except for g are regular elements. We can show that the point, $e\top$, which represents the source of e, is contained in j as follows:

Theorem 2. $e\top \leq j$

Proof.

$$\begin{aligned} e\top &= m(c\overline{c}^{\top} \sqcap g)\top && (3.9)\\ &\leq \overline{\overline{(c\overline{c}^{\top} \sqcap g)}}\top && (3.10)\\ &= (\overline{\overline{c\overline{c}^{\top}}} \sqcap \overline{\overline{g}})\top && (3.11)\\ &= (c\overline{c}^{\top} \sqcap \overline{\overline{g}})\top && (3.12)\\ &= (c \sqcap \overline{c}^{\top} \sqcap \overline{\overline{g}})\top && (3.13)\\ &\leq c\top && (3.14)\\ &\leq j\top && (3.15)\\ &= j && (3.16) \end{aligned}$$

We have (3.9) from the definition of e. Axiom (2.17) gives us (3.10) and we can then simplify to (3.14) because c is a regular vector and due to the results of Theorems 2, 8 and 9 of [42]. Because j is regular, axiom (3.2) of k-Stone relation algebras gives us (3.15) and then, since j is a vector and $\leq$ is transitive, we have $e\top \leq j$. □

The interpretation of this result is that the source of e is a vertex in the set of vertices that are still to be processed by the inner loop, as we would expect.

Chapter 4

Correctness of Borůvka's MST algorithm

In this chapter we discuss the partial-correctness proof of the formalization presented in Chapter 3. We work in m-k-Stone-Kleene relation algebras and our proof holds for any instance of those algebras. In particular, it holds for weighted matrices, $S = \mathbb{R}'^{A \times A}$, representing weighted graphs as discussed in Section 2.3.5.

We begin by giving a high-level overview of how our Hoare-logic proof is structured in Section 4.1.

In Section 4.2 we introduce E-forests, a structure that we use to model and reason about reachability. We also give a result that allows us to compare weights of particular arcs in an E-forest. Edge weight comparison is a crucial aspect of Borůvka's MST algorithm and this result is one of the most important results in our proof.

The specification of a minimum spanning forest is given in Section 4.3. This is the specification that the output of our algorithm must satisfy. In this section we also give the invariants for both the inner and outer loops of our formalization.

In Section 4.4 we discuss how we establish the invariants. We also give three examples of how the invariants are maintained. The first example uses a chain of reasoning to show that the relationship between the forest, f, and the temporary variable from the inner while-loop that is a copy of the forest, h, is maintained as we add arcs to f. The second example uses case distinction to show that, as arcs are added to the forest, the result that allows edge weight comparison between particular arcs in an E-forest continues to hold. Finally, in Section 4.4.6 we discuss how we maintain the invariant that the forest, f, may be extended to a minimum spanning forest.

4.1 Proof overview

We use Hoare-logic to complete our partial-correctness proof of Borůvka's MST algorithm. We enter an annotated version of our formalization from Section 3.2 into Isabelle/HOL. The annotations include the precondition, the invariants of both the inner and outer while-loops, and the postcondition, to be discussed in Section 4.3. We use a Hoare-logic library to generate proof goals [65]. For a while-loop nested in another while-loop it creates five proof goals.

The first goal is to show that if the precondition holds then the outer while-loop invariant also holds.

The next three goals are to show that that if the outer while-loop invariant holds then the inner while-loop invariant can be established, that the inner while-loop invariant is maintained following each iteration of the inner while-loop, and that when the inner while-loop terminates the outer while-loop invariant is maintained.

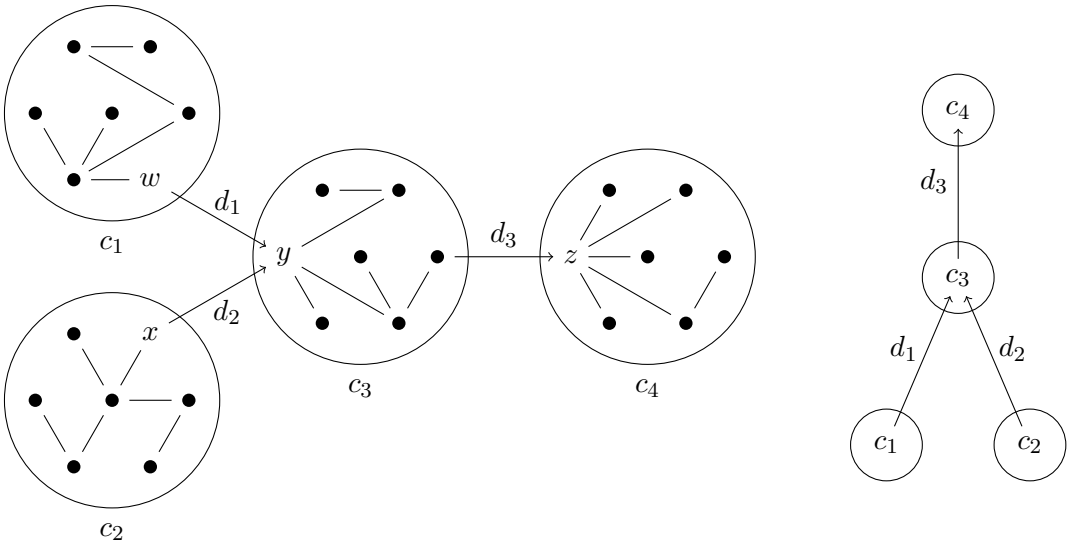


Figure 4.1: On the left, an example E-forest. The arcs in d are labeled d_1 to d_3. The components described by $E = c(h)$ are enclosed in circles and labeled c_1 to c_4. The directions of the arcs within these components are not shown because the components, $c(h)$, form an equivalence that represents any number of steps, in any direction, within a component. A simpler 1-forest representation of this structure is shown on the right.

Finally, the last proof goal is to show that when the outer while-loop exits the postcondition can be established.

Once we have discharged each of these goals we can assume that the postcondition holds. This shows that our formalization produces an element of the m-k-Stone-Kleene relation algebra that satisfies the formal specification of a minimum spanning forest.

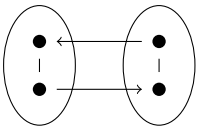
4.2 A reachability structure for forests

Within the inner loop of the algorithm the rooted directed forest is grown by some number of arcs that connect its components. We introduce an abstraction called an *E-forest* that encapsulates the idea of reachability in a structure comprised of the components of a rooted directed forest connected by arcs.

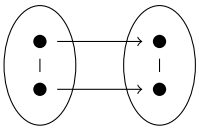
An E-forest, d, is comprised of an equivalence, E, and a set of arcs, d. We are particularly interested in the case $E = c(h)$, that is, where the equivalence is the snapshot of the components of the rooted directed forest, h, as they were at the start of the inner loop. The set of arcs, d, represents those arcs that have been added to connect the components of h since the start of this iteration of the outer loop.

We use the terms *incoming* and *outgoing* to describe an arc with respect to a component. An arc that is incoming to a component describes an arc that has a source outside of that component and a target inside that component. An arc that is outgoing from a component describes an arc that has a source inside that component and a target outside that component. We also say that incoming and outgoing arcs are *adjacent* to a component. For example, in Figure 4.1, arc d_1 is outgoing from component c_1 and incoming to component c_3. Arc d_3 is adjacent to both c_3 and c_4.

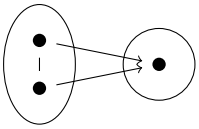
Definition 14. *Let S be a Stone-Kleene relation algebra and let $E, d \in S$ where E is an*



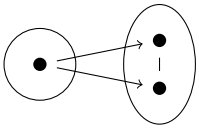
(a) This is not an E-forest because there is a cycle between components, that is, $(Ed)^+ \leq \overline{E}$ does not hold.



(b) This is not an E-forest because there is a component with more than one outgoing arc: $d^\top Ed \leq 1$ is not satisfied.



(c) This is not an E-forest because there is a component with more than one outgoing arc: $E \sqcap dd^\top \leq 1$ is not satisfied.



(d) This is not an E-forest because there is component with more than one outgoing arc: $d^\top Ed \leq 1$ is not satisfied.

Figure 4.2: Four simple examples of structures that do not satisfy the axioms of an E-forest. The components that the equivalence E represents are enclosed in ovals and d is shown as edges between the components.

*equivalence. Then, d is an E-*forest *if the following axioms are satisfied:*

$$d \leq \overline{E} \tag{4.1}$$

$$d^\top Ed \leq 1 \tag{4.2}$$

$$E \sqcap dd^\top \leq 1 \tag{4.3}$$

$$(Ed)^+ \leq \overline{E} \tag{4.4}$$

An example of an E-forest is shown in Figure 4.1. It consists of a number of components, specified by E, and a number of arcs connecting those components, specified by d. If the components are collapsed into a single vertex the resulting graph is a forest, specifically a tree.

It should be the case that all arcs in d do not have both a source and target within the same component. This is expressed by axiom (4.1).

Axiom (4.2) expresses the univalent-like structure of an E-forest, that is, Ed is univalent. This is part of a constraint that ensures each component of h has at most one outgoing arc. For example, in Figure 4.1 consider a sequence of steps allowed by $d^\top c(h)d$ starting from vertex y. The first step backward in d could be along d_1 which would take us to vertex w. We then would take any number of steps, forwards and backwards in component c_1. Finally, the only step allowed forwards in d is again d_1 taking us back to vertex y. This, and any other sequence allowed by $d^\top c(h)d$, results in a loop, which is why the left hand side of axiom (4.2) is below 1. Examples of structures that do not satisfy this axiom can be seen in Figures 4.2(b) and 4.2(d).

Axiom (4.3) is the other constraint that, in conjunction with axiom (4.2), ensures each component has at most one outgoing arc. For example, in Figure 4.1, starting at vertex w then taking a step forwards then a step backwards in d could leave us at vertex x or back at vertex w. By taking the meet with $c(h)$, that is, $c(h) \sqcap dd^\top$, we restrict those arcs to arcs contained in a component. Then, only loops should remain which, in the example given, is the loop on vertex w. Figure 4.2(c) is an example of a structure that does not satisfy this axiom.

Lastly, axiom (4.4) ensures that if we were to take any number of steps in a component followed immediately by a single step between components, one or more times, we will not find ourselves back in the same component. This expresses the acyclic-like structure of the E-forest. For example, in Figure 4.1, it is possible to reach vertex z from vertex x by $c(h)d_2c(h)d_3$ and these vertices are in different components. There is no sequence of steps expressed by $(c(h)d)^+$ that we could take from vertex x where we would arrive in component c_2. An example of a structure that does not satisfy this axiom can be seen in Figure 4.2(a).

The name E-forest arises from the idea that it is a forest-like structure. If the components of the equivalence each contain a single vertex such that $E = 1$ then we call the resulting structure a 1-forest. The axioms from Definition 14 then describe an element that is univalent, $d^\top d \leq 1$, and acyclic, $d^+ \leq \overline{1}$, from axioms (4.2) and (4.4) respectively. Axiom (4.3) is satisfied trivially, and axiom (4.1) follows from axiom (4.4) owing to $d \leq d^+ \leq \overline{1}$. Recall, from Section 2.3.6, that an acyclic and univalent element describes a rooted directed forest, where the arcs are directed towards the root vertices. Therefore, a 1-forest is a rooted directed forest.

Usually, we will talk about a particular instance of an E-forest where the equivalence is the result of the component operation discussed in Section 3.3.1. For example, we would call the E-forest using an equivalence made from the components of forest, h, a $c(h)$-forest.

4.2.1 Properties of E-forests

We have proved a number of properties about the E-forest abstraction. We discuss a selection of these, beginning with properties that apply more generally to equivalences and arcs.

Theorem 3. *Let S be a Stone-Kleene relation algebra. Then, for all $a, E, x \in S$ where a is an arc and E is an equivalence, the following properties hold:*

$$Ea = Ea(\top Ea)^* \tag{4.5}$$

$$(E(x \sqcup a))^* = (Ex)^* \sqcup (Ex)^* Ea (Ex)^* \tag{4.6}$$

The most interesting case of Property (4.5) is where a lies between equivalence classes of E. It says that starting anywhere in an equivalence class, moving to the source of a, and then stepping along a to another equivalence class is the same as doing that and then: moving back to the first equivalence class, moving to the source of a, and stepping along a to another equivalence class, any number of times. Consider, for example, Figure 4.1. The result of the sequence of steps starting from vertex y, moving through the equivalence class representing component c_3, and taking edge d_3 to vertex z is the edge (y, z). This is the same as if after making the step (y, z) we were to jump anywhere in c_3 and move back to z some number of times.

Property (4.6) is a separation rule for the Kleene star. If the arc, a, is contained in x then this becomes trivial. The interesting case is where a is not contained in x. On the left-hand side we can make any number of steps in the equivalence E and then a single step in either x or a. This can be done any number of times. However, because a is an arc, once a step is taken along it it is not necessary to do so again.

We also prove properties about E-forests in particular.

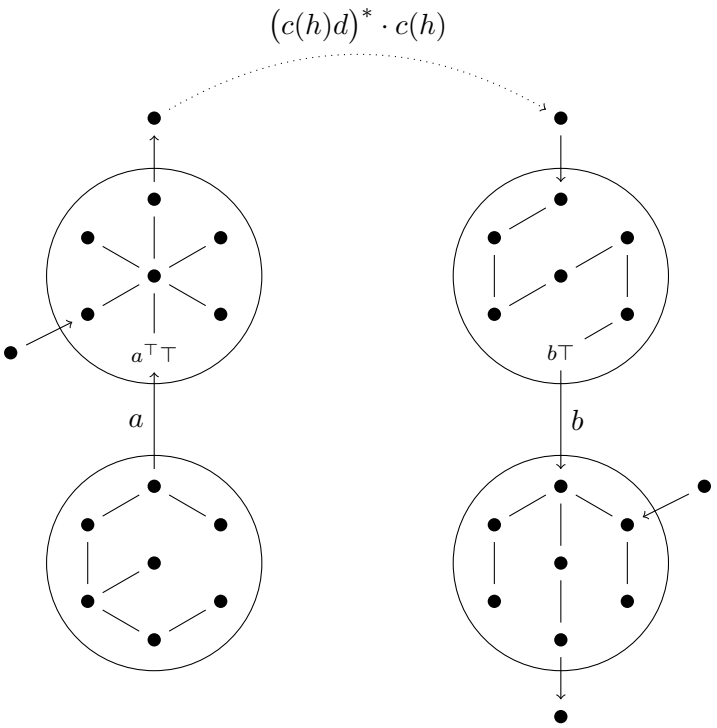
Theorem 4. *Let S be a Stone-Kleene relation algebra. Then, for all $d, E, a \in S$ where E is an equivalence, d is an E-forest, and a is an arc, the following properties hold:*

$$(d^\top E)^* (Ed)^* = (d^\top E)^* \sqcup (Ed)^* \tag{4.7}$$

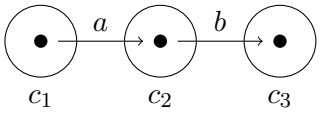
$$a \leq d \Rightarrow (d \sqcap \overline{a})^\top (Ea\top) \leq \bot \tag{4.8}$$

Property (4.7) follows from E being an equivalence and the fact that E-forests are univalent. Notice that $d^\top E$ is the transpose of Ed. This property states that taking any number of steps backwards in the E-forest (away from the roots) followed by any number of steps forwards in the E-forest (towards the roots) is the same as going either forwards or backwards. This is similar to how the components of a forest may be equivalently represented as $(f \sqcup f^\top)^*$ or $f^{\top *} f^*$ owing to a forest's injectivity, as was discussed in Section 3.3.1.

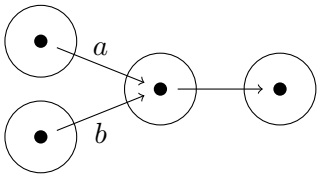
Property (4.8) states that if we take a step backwards between components of an E-forest, without using some arc, a, that lies between components, then there is no sequence of steps we can then take in the component we find ourselves in to then be able to take a step along arc a. For example, starting from vertex y in Figure 4.1 we take a step backwards along d_2, that is, $(d \sqcap \overline{d_1})^\top$, to arrive at vertex x. From there, we can move anywhere in component c_2 but it will not be possible to then take a step along arc d_1.



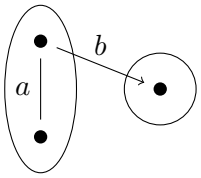
(a) An example of a $c(h)$-forest-path. The target of a, that is, $a^\top\top$, and the source of b, that is, $b\top$, are shown.



(b) The weights of a and b may be directly compared since they are adjacent to the same component.



(c) There is no path from the target of a to the source of b.



(d) The arc a is contained in a component.

Figure 4.3: Examples of $c(h)$-forest-paths, and non-$c(h)$-forest-paths between two arcs, a and b. The components of h are enclosed in ovals. The directions of arcs within components are not shown. A dotted line represents zero or more components, connected by arcs of d.

4.2.2 E-forest paths

Our formalization constructs a $c(h)$-forest in a way that allows us to compare edge weights among certain edges. For example, in Figure 4.3(a) we could show that the weight of b is less than the weight of a. To achieve this, we establish and maintain an invariant, the details of which are discussed in Section 4.2.3.

First we define a general expression for a path between two vertices in an E-forest.

Definition 15. *Let a, b, d, E, and g be elements of an m-k-Stone-Kleene relation algebra. Then, $\langle a\,,\ b\,,\ d\rangle$ is an* E-forest-path *in g if a, b, d and E are regular, a and b are arcs, d is an E-forest, E is an equivalence, and the following axioms are satisfied:*

$$a^\top\top \leq (Ed)^* Eb\top \tag{4.9}$$

$$a \leq \overline{E} \sqcap \overline{\overline{g}} \tag{4.10}$$

$$b \leq d \tag{4.11}$$

Condition (4.9) verifies that there is a path in the E-forest, d, from the target of a to the source of b. The target of a is in the set of predecessors of the source of b. An example of this is shown in Figure 4.3(a). Condition (4.10) ensures that a is in g and is not contained in a component of the E-forest. Lastly, condition (4.11) ensures that b is contained in d.

In the remainder of this thesis we do not explicitly mention the graph, g, that an E-forest-path refers to: it should be understood from context. We abbreviate E-forest-path as $a \rightsquigarrow^d_E b$ or

$$a \overset{d}{\underset{E}{\rightsquigarrow}} b$$

The following result says that there is a path in the E-forest, $d \sqcup e$, between a and b if and only if there is either a path in the E-forest, d, from a to b or one from a to e and one from e to b.

Theorem 5. *Let a, b, E, d and e be elements of an m-k-Stone-Kleene relation algebra where e is an arc. Then,*

$$a \underset{E}{\overset{d \sqcup e}{\rightsquigarrow}} b \iff a \underset{E}{\overset{d}{\rightsquigarrow}} b \vee \left(a \underset{E}{\overset{d}{\rightsquigarrow}} e \wedge e \underset{E}{\overset{d}{\rightsquigarrow}} b \right)$$

Theorem 5 allows us to split E-forest-paths. This is primarily used to make case distinctions. We give an example of this in Section 4.4.5.

4.2.3 Arc weight comparison in $c(h)$-forests

The $c(h)$-forest structure allows a convenient comparison between the weights of the arcs in d. For example, in Figure 4.3(b), the weights of arcs a and b may be compared because they are both adjacent to the same component. Since b is outgoing from c_2, it must have a weight that is less than or equal to all other arcs that are adjacent to that component. This is because, as discussed in Section 3.3.3, when the algorithm was processing c_2, b was chosen as the arc with minimal weight among those that were adjacent to the component. So we can, for example, conclude that the weight of b is less than or equal to the weight of a. By the same argument, a must have a weight less than or equal to any arc that is adjacent to c_1.

This idea is generalized to $c(h)$-forest-paths and we show that for any arcs, a and b, where there is a $c(h)$-forest-path from a to b, the weight of b is less than or equal to the weight of a, that is,

$$a \underset{c(h)}{\overset{d}{\rightsquigarrow}} b \implies s(b \sqcap g) \leq s(a \sqcap g) \tag{4.12}$$

For example, we can use this property to compare the weights of arcs a and b in Figure 4.3(a). We know that a is in the graph and is not contained in a component, b is in d, and there is a path from a to b in the $c(h)$-forest. Hence, we can conclude that the weight of b is less than or equal to the weight of a. It should be apparent that this property does not provide a means to compare all arcs in a $c(h)$-forest. For example, we cannot use the property to compare the arc weights of a and b in Figures 4.3(c) and 4.3(d).

This property does not hold in general for E-forest-paths. We maintain an inner-loop invariant to show that it does hold for $c(h)$-forest-paths that are handled by the inner loop of our formalization.

4.3 Conditions and invariants

In this section, we give the specification of what the output of our algorithm should satisfy and list the invariants for both the inner and outer loops of our formalization.

4.3.1 Specification

Upon termination of the algorithm, it should be the case that the output f is a minimum spanning forest of the input graph, g. To show this, we need a specification for what it is to be a minimum spanning forest, expressed in the algebra that we are working in. To specify a minimum spanning forest we were able to reuse the formal specification from [41, 44] that was created for Kruskal's MST.

Definition 16. *Let S be an m-Kleene algebra where $f, g \in S$. Then, f is a* spanning forest *of g if f is a regular, rooted directed forest and*

$$f \leq \overline{\overline{g}} \tag{4.13}$$

$$\overline{\overline{g}}^* \leq c(f) \tag{4.14}$$

The spanning forest, f, is a minimum spanning forest *of g if for all $u \in S$ where u is a spanning forest of g, the following holds:*

$$s(f \sqcap g) \leq s(u \sqcap g) \tag{4.15}$$

Note that the minimum selection operation of the m-Kleene algebra, axiomatized in (2.17-2.19), is not required for this definition.

Axiom (4.13) ensures that the rooted directed forest f is a subgraph of g, ignoring edge weights. Axiom (4.14) requires that the components of g are contained in the components of the rooted directed forest. As discussed in Section 2.3.5, the double pseudo-complement of the graph, $\overline{\overline{g}}$, removes the weight information and so these axioms talk about the structure of the graph. To specify that the spanning forest is a minimum spanning forest we use the summation operation of m-Kleene algebras, s, in axiom (4.15).

4.3.2 The outer loop

To show that the output of the algorithm, f, satisfies this specification we were able to reuse much of the invariant from [44]. When the outer loop terminates we have enough information to be able to conclude our proof.

The precondition is that g is symmetric. The invariant of the outer loop maintains that

1. g is symmetric,
2. f is a rooted directed forest,
3. $f \leq \overline{\overline{g}}$, meaning that f is contained in g, ignoring arc weight,
4. f is regular, and
5. there is a minimum spanning forest, w, such that $f \leq w \sqcup w^\top$.

Then, we guarantee the postcondition that f is a minimum spanning forest of g.

The invariant of the outer loop is similar to the invariant of Kruskal's MST algorithm described in [44], with some differences. Our invariant does not mention a variable tracking processed arcs because, as described in Section 2.2, while Kruskal's MST algorithm iterates over the arcs of the graph, Borůvka's MST algorithm does not. We also do not require that f is a spanning forest of g, ignoring unprocessed arcs. This invariant is used for the establishment of the postcondition in the proof of Kruskal's MST algorithm but in our case it follows from other properties. Namely, we obtain that f is a regular, rooted directed forest and $f \leq \overline{\overline{g}}$ directly from the invariant. From the negation of the outer loop condition, we can show axiom (4.14)

$$\begin{aligned}
& \overline{c(f)} \sqcap g = \bot \\
\Leftrightarrow\ & \overline{c(f)} \sqcap g \leq \bot \\
\Leftrightarrow\ & g \leq \overline{\overline{c(f)}} \\
\Rightarrow\ & \overline{\overline{g}} \leq c(f) \\
\Rightarrow\ & \overline{\overline{g}}^* \leq c(f)
\end{aligned}$$

Therefore f is a spanning forest of g. To get that f is a minimum spanning forest we use the last part of the invariant of the outer loop, that there exists a minimum spanning forest, w, such that $f \leq w \sqcup w^\top$ and derive

$$s(f \sqcap g) = s(w \sqcap g)$$

Since w is a minimum spanning forest of g, axiom (4.15) is satisfied and all proof obligations have been discharged.

4.3.3 The inner loop

To support showing that the invariant of the outer loop holds, our inner-loop invariant has many properties. The invariant of the inner loop maintains that:

1. the invariants of the outer loop also hold,
2. $g \neq \bot$, meaning that the graph has at least one arc,
3. j is a vector,
4. j is regular,
5. h is a rooted directed forest,
6. $h \leq \overline{\overline{g}}$, meaning that h is contained in g, ignoring arc weights,
7. h is regular,
8. there is a minimum spanning forest, w, such that $h \leq w \sqcup w^\top$,
9. $c(h) \leq c(f)$, meaning that the components of h are contained in the components of f,
10. d is a $c(h)$-forest, that is, the components of h connected by the arcs in d form a $c(h)$-forest,
11. $d\top \leq \overline{j}$, meaning that the sources of the arcs in d are not in the set of vertices still to be processed,
12. $c(h)j = j$, meaning that j contains each component of h entirely or not at all,
13. $c(f) = \big(c(h)\ (d \sqcup d^\top)\big)^* c(h)$, meaning that the components of f can be obtained by taking any number of steps in the $c(h)$-forest, ignoring arc direction,
14. $f \sqcup f^\top = h \sqcup h^\top \sqcup d \sqcup d^\top$, meaning that, ignoring direction, f can be obtained by taking the join of h and d,
15. $\forall a, b : a \rightsquigarrow^d_{c(h)} b \implies s(b \sqcap g) \leq s(a \sqcap g)$, meaning that, for any arcs a and b, if there is a $c(h)$-forest-path from a to b then the weight of b is less than or equal to the weight of a, and
16. d is regular.

Invariant 15 is particularly important for our proof and we discuss the maintenance of this invariant in detail in Section 4.4.3.

4.4 Proof

We are performing a Hoare-logic proof so the most challenging part of our work was to choose appropriate loop invariants and then maintain them. Aside from variable initialization, most of the logic of our formalization is found inside the inner loop. This makes the maintenance of the inner loop invariant particularly difficult. In this section, we give examples of how we established and maintained the inner and outer invariants that were introduced in Sections 4.3.3 and 4.3.2.

Before we discuss more interesting examples from our proof we give a selection of results we have proved that are more general.

4.4.1 A selection of general results

In addition to the results that are specialized toward a weighted-graph instance of m-k-Stone-Kleene relation algebras, we have proved some more general results. In the following theorem, we present a selection of these results.

Theorem 6. *Let S be an m-Kleene algebra. Then, for all $a, b, x, y \in S$, where a and b are arcs*

$$yx^*y \leq y \Rightarrow (x \sqcup y)^* = x^* \sqcup x^*yx^* \tag{4.16}$$

$$a = aa^\top a \tag{4.17}$$

$$a \leq x \sqcup b \Rightarrow a \leq x \vee a \leq b \tag{4.18}$$

$$\neg(a \leq b) \Rightarrow a \leq \overline{b} \tag{4.19}$$

Property (4.16) is a separation rule for the Kleene star. The condition $yx^*y \leq y$ also implies $(yx^*)^*y \leq y$ from the induction axioms. We use this property when reasoning about paths through components.

Property (4.17) states that taking a single step along an arc, then back, then along the arc again is equivalent to taking a single step along the arc.

Property (4.18) states that if an arc, a, is contained in the join of an arc, b, and some other element, x, then it must be contained either in x or in b.

Property (4.19) states that if an arc, a, is not contained in an arc, b, then it must be in b's complement. The implication could also be written as $a \leq b \vee a \leq \overline{b}$.

4.4.2 Establishing invariants

To establish the outer invariant we were able to reuse lemma *kruskal-exists-minimal-spanning* by Guttmann [41]. This lemma shows that a symmetric graph has a minimum spanning forest. Since we assume our input graph is symmetric and we initialize f to be $\bot$ we can immediately establish the outer-loop invariant as follows:

- g is symmetric follows from the precondition,
- f is regular since $\bot$ is regular,
- f is a rooted directed forest since $\bot$ is injective, $\bot\bot^\top \leq 1$, and acyclic, $\bot^+ \leq \overline{1}$,
- f is contained in g ignoring arc weight since $f = \bot \leq \overline{\overline{g}}$, and
- there is a minimum spanning forest w such that $f \leq w \sqcup w^\top$ owing to the precondition and the above-mentioned lemma.

Establishing the invariant of the inner loop was similarly easy. For example, the graph is not empty, that is, $g \neq \bot$, immediately follows from the condition of the outer while-loop, $\overline{c(f)} \sqcap g \neq \bot$. Another condition that must be established is that the components of f can be obtained by taking any number of steps in the $c(h)$-forest, ignoring arc direction, that is, $c(f) = \big(c(h)\ (d \sqcup d^\top)\big)^* c(h)$. Because d is initialized as $\bot$ and h is initialized as f in the inner while-loop, it follows that

$$\begin{aligned}(c(h)\ (d \sqcup d^\top))^* c(h) &= \big(c(h)\bot\big)^* c(h)\\ &= \bot^* c(h)\\ &= c(h)\\ &= c(f)\end{aligned}$$

Sledgehammer was able to find proofs that established the invariants of both our inner and outer loops.

4.4.3 Maintaining invariants

Since g is not updated, anything that we establish about g will follow immediately, in particular, that $g \neq \bot$. The maintenance of other invariants is not so trivial. In the following sections we discuss two examples of maintaining the inner loop invariant. Firstly, f can be obtained by taking any number of steps in the $c(h)$-forest, ignoring arc direction. Secondly, for any arcs, a and b, where there is a $c(h)$-forest-path from a to b the weight of b is less than or equal to the weight of a. We then discuss how we maintain the outer loop invariant that the rooted directed forest, f, can be extended to a minimum spanning forest of the graph.

4.4.4 Maintaining the relationship between f and the $c(h)$-forest

To maintain the invariant that f can be obtained by taking any number of steps in the $c(h)$-forest, ignoring arc direction, we show that this property still holds when f and d are updated in the inner loop. We assume that $c(f) = (c(h)\ (d \sqcup d^\top))^* c(h)$ and then show that

$$c(f') = \big(c(h)\ (d' \sqcup d'^\top)\big)^* c(h) \tag{4.20}$$

where

$$f' = \Big(f \sqcap \overline{e}^\top \sqcap \overline{\top e\big(f \sqcap \overline{e}^\top\big)^{\top *}}\Big) \sqcup \Big(f \sqcap \overline{e}^\top \sqcap \top e\big(f \sqcap \overline{e}^\top\big)^{\top *}\Big)^\top \sqcup e$$

which is the update of the forest discussed in Section 3.3.4, and where

$$d' = d \sqcup e$$

as listed in line 13 of the algorithm in Figure 3.2.

It is often more difficult to algebraically manipulate an expression containing composition than join. This is certainly the case for the left-hand side of Equation (4.20), $c(f')$, that is, $f'^{\top *} f'^*$. Since f' is very complex, and $c(f')$ even more so, we use the following result [45] to rewrite our problem.

Theorem 7. *Let S be a Stone-Kleene relation algebra and let $x \in S$ be injective. Then*

$$c(x) = (x \sqcup x^\top)^*$$

We know that all variables in equation (4.20) are regular and that f and f' are injective since they are forests. Then, we use Theorem 7 to rewrite $c(f')$ as $(f' \sqcup f'^\top)^*$. Since we are taking the symmetric closure, we can simplify to $(f \sqcup f^\top \sqcup e \sqcup e^\top)^*$ because, if direction is ignored, then

we ignore the path reversal part of the update to f and just consider the addition of arc e. The following chain of equalities is then applied to show Equation (4.20).

$$\begin{aligned} c(f') &= (f \sqcup f^\top \sqcup e \sqcup e^\top)^* && (4.21)\\ &= (h \sqcup h^\top \sqcup d \sqcup d^\top \sqcup e \sqcup e^\top)^* && (4.22)\\ &= (h \sqcup h^\top \sqcup d' \sqcup d'^\top)^* && (4.23)\\ &= \big((h \sqcup h^\top)^* \, (d' \sqcup d'^\top)\big)^* (h \sqcup h^\top)^* && (4.24)\\ &= \big(c(h) \, (d' \sqcup d'^\top)\big)^* c(h) && (4.25) \end{aligned}$$

We have (4.22) from $f \sqcup f^\top = h \sqcup h^\top \sqcup d \sqcup d^\top$, which is maintained by the inner invariant. Because join is associative and commutative, and using the definition of d', we obtain (4.23). Owing to the sumstar property of * we have (4.24) and then (4.25) follows from Theorem 7.

4.4.5 Maintaining arc weight comparison in a $c(h)$-forest

One key part of the invariant of the inner loop that must be maintained is that for any arcs, a and b, where there is a $c(h)$-forest-path from a to b the weight of b is less than or equal to the weight of a.

We start with the assumption that the invariant holds for the previous loop and additionally assume that there is a $c(h)$-forest-path from a to b in d'. The property of the invariant that we are most interested in is that, for all arcs a, b

$$a \overset{d}{\underset{c(h)}{\rightsquigarrow}} b \implies s(b \sqcap g) \leq s(a \sqcap g) \tag{4.26}$$

We need to show that $a \rightsquigarrow^{d'}_{c(h)} b \implies s(b \sqcap g) \leq s(a \sqcap g)$ for any arcs a, b. To this end, we assume $a \rightsquigarrow^{d'}_{c(h)} b$, that is, a, b, $c(h)$ and d' are regular, a and b are arcs, $c(h)$ is an equivalence, d' is a $c(h)$-forest and furthermore, that

$$\begin{aligned} & a^\top \top \leq \big(c(h) \cdot d'\big)^* \cdot c(h) \cdot b\top \\ \wedge \quad & a \leq \overline{c(h)} \sqcap \overline{\overline{g}} \\ \wedge \quad & b \leq d' \end{aligned}$$

and then show that $s(b \sqcap g) \leq s(a \sqcap g)$.

The proof that this invariant is maintained as d is updated is performed by case distinctions. The case distinctions, along with example graph structures for each of the cases, are shown in Figure 4.4. We first make a case distinction on whether $b = e$ or not.

When $b \neq e$ we make an additional distinction on whether e is contained in d or not. If $e \not\leq d$ then we use Theorem 5 to split this case into cases (1) and (2) from Figure 4.4. Case (3) is when e is contained in d.

When $b = e$ we make an additional distinction on whether $a = e$ or not. If $a \neq e$, we make a further case distinction on whether a is incoming to the component that b is outgoing from or not. The case where it is not, that is, $a^\top \top \not\leq c(h)e\top$, is case (4) shown in Figure 4.4. Case (5) shows where a is incoming to the component that b is outgoing from. Finally, case (6) in Figure 4.4 is where $b = e$ and $a = e$.

Cases (1), (3) and (6) can be shown immediately while the remaining cases require more work to prove.

Case (1) In this case there is a $c(h)$-forest-path from a to b without using e as an intermediate arc, therefore, we have all the conditions required to show that $s(b \sqcap g) \leq s(a \sqcap g)$ using assumption (4.26).

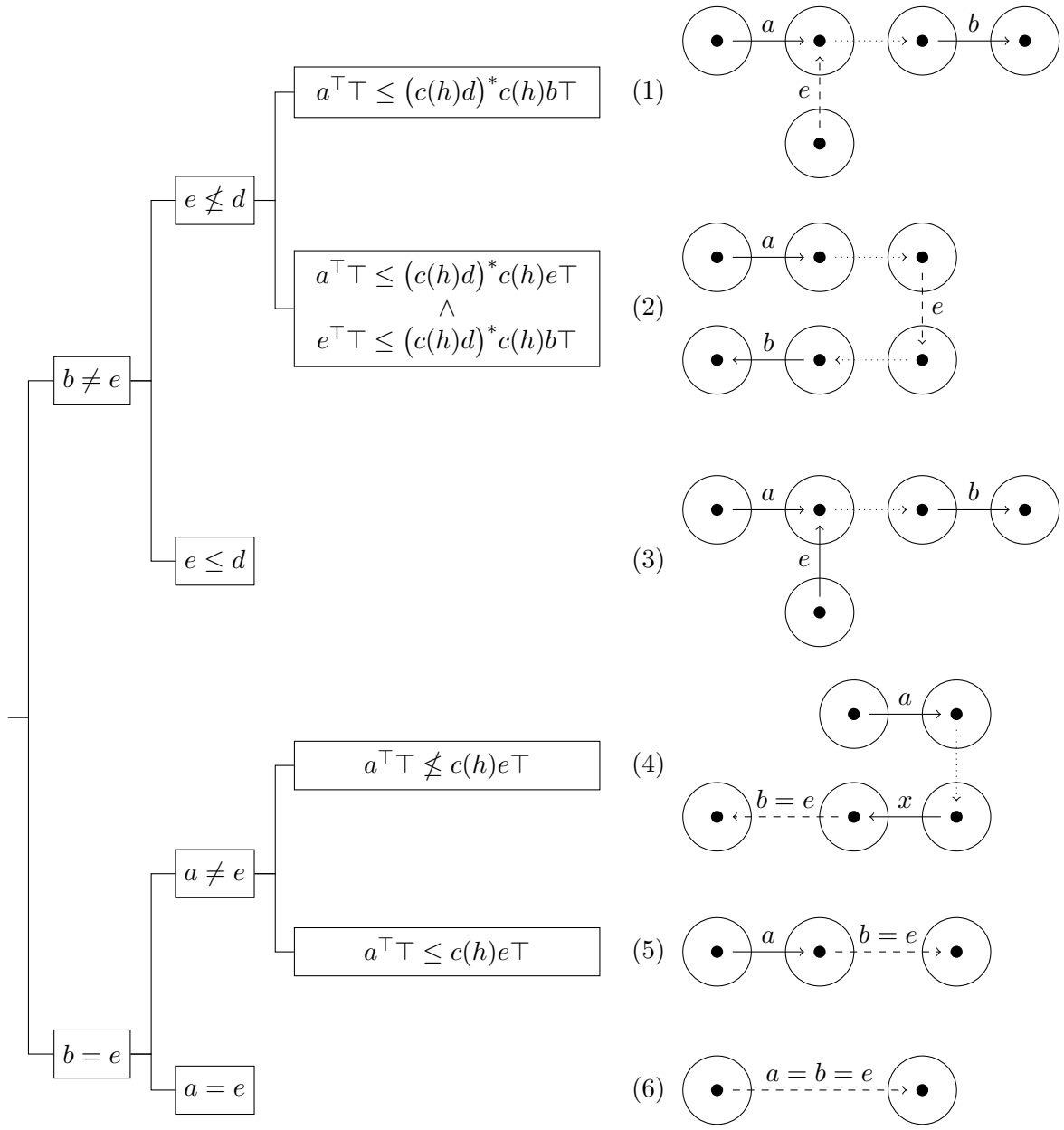


Figure 4.4: Six case distinctions to maintain the inner loop invariant that for any arcs, a and b, where there is a $c(h)$-forest-path from a to b the weight of b is less than or equal to the weight of a. Each of the cases is defined by the conjunction of the expressions in the left tree. For example, case (6) is where $b = e$ and $a = e$. On the right side of the figure are examples of what these cases look like in the graph. A dotted line indicates zero or more components connected by arcs in d. A dashed line denotes an arc that is not in d.

Case (2) In this case we have that $b \neq e$ and $e \not\leq d$ and $a^\top\top \leq \big(c(h)d\big)^* c(h)e\top$, and $e^\top\top \leq \big(c(h)d\big)^* c(h)b\top$.

First we consider the path from a to e, that is, $a^\top\top \leq \big(c(h)d\big)^* c(h)e\top$ and recognize that we can conclude that $s(e \sqcap g) \leq s(a \sqcap g)$ by applying the same logic from cases (4) and (5).

Next, we consider the path from e to b. Since e is contained in the graph, and is not contained in a component of $c(h)$ then $e \leq \overline{g} \sqcap \overline{c(h)}$. Also, because b is an arc, $b \leq d \sqcup e$ and $b \neq e$ then we have $b \leq d$. Therefore, we have $b \rightsquigarrow^d_{c(h)} e$ so it follows from assumption (4.26) that $s(b \sqcap g) \leq s(e \sqcap g)$.

Because $s(e \sqcap g) \leq s(a \sqcap g)$ and $s(b \sqcap g) \leq s(e \sqcap g)$, we obtain our desired result $s(b \sqcap g) \leq s(a \sqcap g)$.

Case (3) In this case $b \neq e$ and $e \leq d$. Because $e \leq d$, it follows that $a \rightsquigarrow^{d \sqcup e}_{c(h)} b$ if and only if $a \rightsquigarrow^{d}_{c(h)} b$. Therefore, we can conclude that $s(b \sqcap g) \leq s(a \sqcap g)$ owing to assumption (4.26).

Case (4) In this case $b = e$ and $a \neq e$ and $a^\top \top \not\leq c(h) e \top$. We start by identifying an arc, x that is incoming to the component that b is outgoing from and which is also in the $c(h)$-forest, as shown in Figure 4.4, case (4). The arc x is defined as

$$x = d \sqcap \top e^\top c(h) \sqcap \big(c(h) d^\top\big)^* c(h) a^\top \top$$

The meet with d ensures that x is an arc between components of the $c(h)$-forest. The second part of this expression, $\top e^\top c(h)$, ensures that the target of x is in the same component of $c(h)$ as the source of e. The last part of this expression, $\big(c(h)d^\top\big)^* c(h) a^\top \top$, ensures that the source of x is reachable from the target of a by taking any number of steps in the $c(h)$-forest. The expression is used to show that x is regular, $x \leq \overline{c(h)} \sqcap \overline{\overline{g}}$, and $x^\top \top \leq c(h) e \top$. We also show that x is an arc which is the part of our proof that requires the Tarski rule, (3.8) of Definition 13, to show that $\top x \top = \top$.

Then we use Theorem 11, discussed later, to show that $s(e \sqcap g) \leq s(x \sqcap g)$.

We prove that $s(x \sqcap g) \leq s(a \sqcap g)$ by showing that the conditions of Definition 15 are satisfied.

$$a \overset{d}{\underset{E}{\rightsquigarrow}} x \implies s(x \sqcap g) \leq s(a \sqcap g)$$

Then we conclude that $s(b \sqcap g) \leq s(a \sqcap g)$ since

$$\begin{aligned} s(b \sqcap g) &= s(e \sqcap g) \\ &\leq s(x \sqcap g) \\ &\leq s(a \sqcap g) \end{aligned}$$

Case (5) In this case $b = e$ and $a \neq e$ and $a^\top \top \leq c(h) e \top$. We show that a is a regular arc, is contained in the graph and is not contained within any component of $c(h)$. We have all the assumptions required to use Theorem 11, presented later, so we can conclude that $s(b \sqcap g) \leq s(a \sqcap g)$.

Case (6) The final case is where $b = e$ and $a = e$, as shown in Figure 4.4. Since $a = b$ we have $s(b \sqcap g) \leq s(a \sqcap g)$.

In the remainder of this section we present Theorem 11. This theorem allows us to show that the selected arc, e, that is outgoing from a component must have a weight less than or equal to any other arc incoming to that component in the $c(h)$-forest. We first present supporting results that are used by Theorem 11. Like all results in this thesis, they are formally verified in Isabelle/HOL.

The following result shows that the source of e is contained in the component c and the target of e is not contained in c, that is, e is outgoing from component c.

Theorem 8. *Let S be a m-k-Stone-Kleene relation algebra and let $c, e, g \in S$ where $e = m(c\overline{c}^\top \sqcap g)$ and c is regular. Then, $e \leq c\overline{c}^\top$.*

Proof.

$$\begin{aligned} e &= m\big(c\overline{c}^\top \sqcap g\big) && (4.27)\\ &\leq \overline{\overline{c\overline{c}^\top \sqcap g}} && (4.28)\\ &= \overline{\overline{c\overline{c}^\top}} \sqcap \overline{\overline{g}} && (4.29)\\ &\leq \overline{\overline{c\overline{c}^\top}} && (4.30)\\ &= c\overline{c}^\top && (4.31) \end{aligned}$$

We have (4.27) from the assumptions. Then, (4.28) follows from axiom (2.17). We then simplify owing to the results of Theorems 2, 8, and 9 of [42] and can remove the double complement (4.31) since c is regular. □

The co-vector $c^\top$ are all arcs that terminate at the component represented by the vector c. The following result shows that the arc x terminates at component c.

Theorem 9. *Let S be a m-k-Stone-Kleene relation algebra and let $x, c, e, h \in S$ where $x^\top\top \leq c(h)e\top$ and c is a vector and $c = c(h)c$ and $e \leq c\overline{c}^\top$ and h is a forest. Then, $x \leq c^\top$.*

Proof.

$$\begin{aligned} x &\leq \top x && (4.32)\\ &\leq \top e^\top c(h) && (4.33)\\ &\leq \top(c\overline{c}^\top)^\top c(h) && (4.34)\\ &= \top\overline{c}c^\top c(h) && (4.35)\\ &\leq c^\top c(h) && (4.36)\\ &= c^\top && (4.37) \end{aligned}$$

We apply the transpose to the assumption, $x^\top\top \leq c(h)e\top$, so that (4.33) follows from Definition 7 and Theorem 8 of [42] and since $c(h)$ is symmetric. We have (4.34) from the assumption that $e \leq c\overline{c}^\top$. We have (4.35) again from Definition 7 and Theorem 8 of [42]. Since c is a vector (4.35) can be simplified to (4.36) by $\top\overline{c}c^\top c(h) \leq \top\top c^\top c(h) = \top c^\top c(h) = c^\top c(h)$. Finally, (4.37) follows because $c^\top c(h) = (c(h)c)^\top = c^\top$, from the assumption $c = c(h)c$ and the fact that $c(h)$ is an equivalence, therefore, symmetric. □

The following result is used to show that the arc x is not contained in the selected component.

Theorem 10. *Let S be a m-k-Stone-Kleene relation algebra and let $x, c, h \in S$ where $x \leq c^\top$ and $x \leq \overline{c(h)}$ and c is a vector and $cc^\top \leq c(h)$. Then, $x \leq \overline{c}$.*

Proof.

$$\begin{aligned} c \sqcap c^\top &= cc^\top && (4.38)\\ &\leq c(h) && (4.39) \end{aligned}$$

then, since $c \sqcap c^\top \leq c(h) \leq \overline{\overline{c(h)}}$, we have

$$\begin{aligned} &\overline{c(h)} \sqcap c \sqcap c^\top \leq \bot && (4.40)\\ \Leftrightarrow\quad &\overline{c(h)} \sqcap c^\top \leq \overline{c} && (4.41) \end{aligned}$$

and finally, it follows that

$$\begin{aligned} x &\leq \overline{c(h)} \sqcap c^\top && (4.42)\\ &\leq \overline{c} && (4.43) \end{aligned}$$

We have (4.38) owing to c being a vector and (4.39) then follows from the assumptions. Next, (4.40) and (4.41) follow from the weak shunting property of Theorem 4 from [42]. Since the assumptions, $x \leq \overline{c(h)}$ and $x \leq c^\top$, imply (4.42) we have (4.43) because of (4.41). □

The following result allows us to compare weights of two arcs under certain conditions. The intuition is, if there is an arc incoming to a component of $c(h)$ in a $c(h)$-forest, and another arc outgoing from that same component, we can show that the weight of the outgoing arc is less than or equal to that of the incoming arc.

Theorem 11. *Let S be a m-k-Stone-Kleene relation algebra and let $x, c, e, h, g \in S$ where x is an arc and $x^\top\top \leq c(h)e\top$ and $x \leq \overline{c(h)} \sqcap \overline{\overline{g}}$ and c is a regular vector and $c = c(h)c$ and $cc^\top \leq c(h)$ and $c \neq \bot$ and $e = m(c\overline{c^\top} \sqcap g)$ and h is a forest and g is symmetric. Then, $s(e \sqcap g) \leq s(x \sqcap g)$.*

Proof.

$$x \leq \overline{c} \sqcap c^\top \tag{4.44}$$
$$= \overline{c}c^\top \tag{4.45}$$

then we apply the transpose

$$x^\top \leq c\overline{c}^\top \tag{4.46}$$
$$\Rightarrow \quad x^\top \sqcap c\overline{c}^\top \sqcap \overline{\overline{g}} \neq \bot \tag{4.47}$$
$$\Rightarrow \quad x^\top \sqcap c\overline{c}^\top \sqcap g \neq \bot \tag{4.48}$$

then, since x is an arc, $x^\top$ is also an arc. It follows that

$$s\big(m(c\overline{c}^\top \sqcap g) \sqcap c\overline{c}^\top \sqcap g\big) \leq s(x^\top \sqcap c\overline{c}^\top \sqcap g) \tag{4.49}$$
$$\Leftrightarrow \quad s\big(e \sqcap c\overline{c}^\top \sqcap g\big) \leq s(x^\top \sqcap c\overline{c}^\top \sqcap g) \tag{4.50}$$
$$\Rightarrow \quad s(e \sqcap g) \leq s(x^\top \sqcap g) \tag{4.51}$$
$$\Leftrightarrow \quad s(e \sqcap g) \leq s(x \sqcap g) \tag{4.52}$$

We have (4.44) from the assumptions and Theorems 8, 9, and 10. The properties of vectors then give (4.45). We apply the transpose to get (4.46). Since x is an arc that is contained in g and because g is symmetric, $x^\top \leq \overline{\overline{g}}$, then we have (4.47). Because x is an arc and given (4.48) then we get (4.49) by axiom (2.19). We can use the definition of e and that c is regular to simplify to (4.51). Finally, axiom (2.14) can be used to show (4.52). □

4.4.6 Extending f to a minimum spanning forest

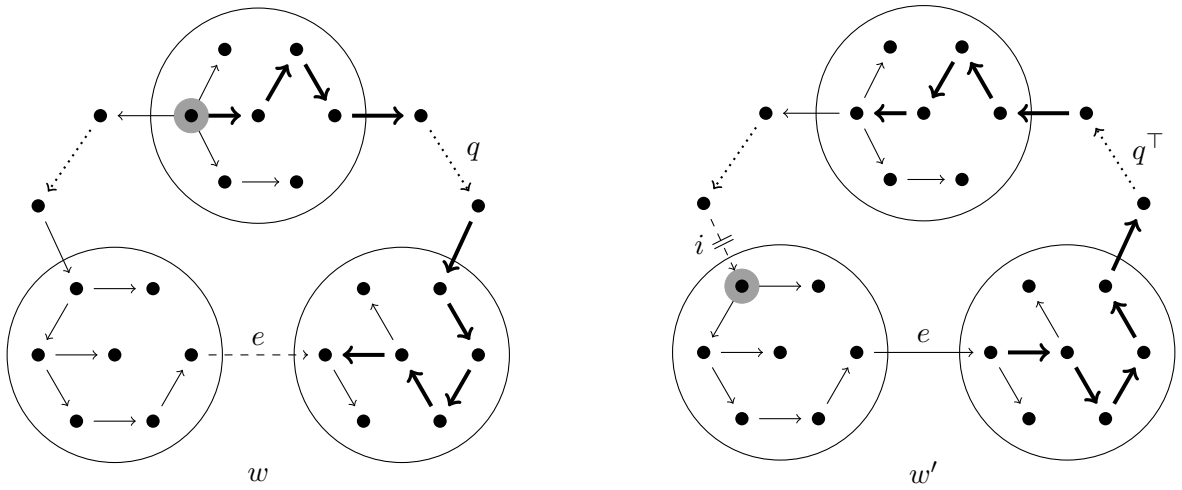
The key property of the invariant of the outer loop that must be maintained is that the rooted directed forest, f, can be extended to a minimum spanning forest of the graph, g, ignoring arc direction, that is, there exists a minimum spanning forest, w, such that $f \leq w \sqcup w^\top$. We were able to reuse some work from [44] in the maintenance of this invariant. However, while the basic structure of the proof of the maintenance of this invariant remains the same, considerable reworking was required.

To maintain this invariant, we assume that $f \leq w \sqcup w^\top$ and then must show that there exists a minimum spanning forest, w', such that, when f is updated to f', we have that $f' \leq w' \sqcup w'^\top$. An illustration how this is done is given in Figure 4.5. In Figure 4.5(a), we see a forest, w, that extends f. In Figure 4.5(b) we see a forest, w', that extends f' in a manner that ensures we can show our invariant is maintained.

When f is updated to f' with the addition of e we must maintain that the minimum spanning forest, w', that extends f', is injective, acyclic, and has weight at most as large as w. To do this we consider some transformations of w and then prove that the properties of interest are maintained.

Firstly, to maintain injectivity, we define a minimum spanning forest, v, in terms of w where any path from the root of w to the target of e is reversed. This is shown as the reversal of q, in Figure 4.5(a), to $q^\top$, in Figure 4.5(b). The path from the root of w to the target of e is

$$q = w \sqcap \top e w^{\top *}$$



(a) A minimum spanning forest, w, that extends f.

(b) A minimum spanning forest, w', that extends f'.

Figure 4.5: Maintaining the invariant that f can be extended to a minimum spanning forest, w, before and after adding arc, e. The path, q, to the root of the rooted directed forest is reversed to maintain injectivity. The arc, i, whose target is in the same component of f as the source of e, is removed to maintain that w' is acyclic. The vertices enclosed in a circle denote a component, in f. The root of the rooted directed forest is highlighted gray.

Then, similar to how we maintain the injectivity of f as discussed in Section 3.3.4, we define v as

$$v = (w \sqcap \overline{q}) \sqcup q^{\top}$$

Secondly, we require that w' is acyclic. If the arc added to f was not also in w then the definition of the minimum spanning forest extending f' must change to ensure that it remains acyclic. This is done by selecting another arc in v and defining w' with that arc removed and e added. Furthermore, we show that this swap results in a spanning tree with weight at most as large as w.

In [44] the arc selected for removal was the arc whose source was in the same component of f as the target of e. This arc does not suit our purposes because we do not have a convenient way to compare the weight of that arc with the weight of e. However, there is an easy comparison to be made between the arc, i, whose target is in the same component of f as the source of e. Namely, the weight of e is at least as small as the weight of i, since i is among those arcs that the algorithm chose e from with the minimum selection $m(c\overline{c}^{\top} \sqcap g)$. The arc i is defined as

$$i = v \sqcap \overline{c(f)}e\top \sqcap \top e^{\top}c(f)$$

The meet with v limits i to only those arcs in the rooted directed forest w, with the path, q, from the root of w to the target of e reversed. The second part of this expression, $\overline{c(f)}e\top$, specifies that the source of i cannot be in the same component of f as the source of e. Finally, the last part of the expression, $\top e^{\top}c(f)$, requires that the target of i is in the same component of f as the source of e. We prove that these requirements uniquely identify an arc, i. After the update, the target of i becomes the root of w' in the component that e is in. Furthermore, we show that $s(e \sqcap g) \leq s(i \sqcap g)$ using Theorem 11.

Therefore, the desired forest, w', that extends f' is v, with i removed and e added, that is,

$$w' = (v \sqcap \overline{i}) \sqcup e$$

This is the construction shown in Figure 4.5(b).

Chapter 5

Conclusion

Our aim was to provide a machine-verified formal partial-correctness proof for Borůvka's MST algorithm. We have given a formal description of Borůvka's MST algorithm using m-k-Stone-Kleene relation algebras. We have completed a formal, partial correctness proof to show that this description satisfies a formal specification for computing minimum spanning forests. The proof has been automatically verified by Isabelle/HOL.

5.1 Limitations and future work

A minor change that could be made to our proof would be to define an E-forest in terms that are closer to the way that forests are defined. In Definition 14, we describe an E-forest as being univalent. This description of a forest is one where the arcs are directed towards the root vertices. However, as discussed in Section 2.3.6, we define a forest as being injective, that is, the arcs of the forest are directed away from the root vertices. It should not require too much work to adjust this definition though a number of theorems of the proof would need to be changed. Making this change would result in a more consistent approach to forest definition across the proof.

We do not prove that the algorithm terminates. Rather, our Hoare-logic proof concludes that if the algorithm terminates then the output is a minimum spanning forest of the input graph. We do not expect this to require a substantial amount of time to complete, in particular because of the prior work by Guttmann to extend the Hoare-logic library we are using to allow for total-correctness proofs.

We claim that our formalization, presented in Section 3.2, is an accurate representation of Borůvka's MST algorithm, with the exception of an additional conditional statement in the inner while-loop, as discussed in the following paragraph. This claim is based on informal reasoning only so is not made with the same confidence as our partial-correctness proof.

We do not give a specification for the input graph to have distinct arc weights. As discussed in Section 2.2.2, this could result in a cycle being created in the algorithm's output. Our formalization circumvents this problem by having a condition in the inner while-loop that checks whether the addition of the arc e would create a cycle in the forest f if it were added. It does this by checking that e is not contained in any component of f and performing a skip operation if it is.

Recall, from Section 1.2, that one motivation for using Borůvka's MST algorithm is the performance gain to be had by leaning on how readily it may be parallelized. One limitation with our approach of using a condition in the inner while-loop that depends on the state of the forest is that it results in a description which is not amenable to parallelization. At least, in practice this would require some synchronization before performing the conditional check.

We suspect that if a specification were given that required the input graph to have only distinct arc weights then we could derive that the selected arc is not contained in a component

of the forest in each iteration of the inner while-loop. We could then remove the condition from the formalization.

5.2 Discussion

We have benefited greatly from the prior work of Guttmann, both from the algebraic framework that we extend and from the theorems and lemmas published in the Archive of Formal Proofs. We have found that Stone-Kleene relation algebras are a useful algebraic framework to prove most of our results. Some parts of our proof required additional structure. There were sections of our proof that additionally required the axioms of m-Kleene relation algebras, in particular for selecting an arc with minimal weight and for comparing weights between arcs.

We extended Stone relation algebras to k-Stone relation algebras with the addition of a component selection operation, k. This operation was not strictly necessary to complete our proof and we could have used an algebraic expression in m-Kleene relation algebras to formalize the selection of a component. However, we found the addition of the k operation for this purpose to more clearly communicate the desired intent to a reader of the formalization.

While most of our proof used only the axioms of Stone-Kleene relation algebras, we work in m-k-Stone-Kleene relation algebras to have access to the component selection operation and because we required the Tarski rule to prove that a particular element is an arc.

Because our proof is conducted using only the axioms of m-k-Stone-Kleene relation algebras, the proof will hold for instances other than the weighted-graph model. We do not explore this further here but note that in [38] it is discussed how different instances of the m-Stone algebras give rise to formalizations of various other algorithms, for example, the minimum bottleneck spanning tree problem. The proof holds for any instance that satisfies the axioms the proof is conducted in. This means that Borůvka's MST algorithm is correct for various related MST problems.

We benefited from the tools and libraries of Isabelle/HOL. Sledgehammer was able to find proofs for many of the smaller goals for us which alleviated the burden of having to know the name and content of each relevant property in the library of theory files. We used the Hoare-logic verification generator library to generate the proof goals for us directly from our formalization. This meant that we did not have to manually generate our verification conditions and, given that we rewrote our formalization and loop invariants a number of times, saved us considerable time.

Bibliography

[1] Jean-Raymond Abrial, Dominique Cansell, and Dominique Méry. Formal derivation of spanning trees algorithms. In *International Conference of B and Z Users*, pages 457–476. Springer, 2003.

[2] Raymond Balbes and Philip Dwinger. *Distributive Lattices*. University of Missouri Press, 1974.

[3] Ralf Behnke, Rudolf Berghammer, Erich Meyer, and Peter Schneider. RelView—a system for calculating with relations and relational programming. In *International Conference on Fundamental Approaches to Software Engineering*, pages 318–321. Springer, 1998.

[4] Rudolf Berghammer and Frank Neumann. RelView–an OBDD-based computer algebra system for relations. In *International Workshop on Computer Algebra in Scientific Computing*, pages 40–51. Springer, 2005.

[5] Rudolf Berghammer, Agnieszka Rusinowska, and Harrie De Swart. Computing tournament solutions using relation algebra and RelView. *European Journal of Operational Research*, 226(3):636–645, 2013.

[6] Rudolf Berghammer, Burghard von Karger, and Andreas Wolf. Relation-algebraic derivation of spanning tree algorithms. In Johan Jeuring, editor, *Mathematics of Program Construction*, volume 1422 of *Lecture Notes in Computer Science*, pages 23–43. Springer, 1998.

[7] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development - Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer, 2004.

[8] Garrett Birkhoff. *Lattice Theory*, volume 25. American Mathematical Society, 1948.

[9] Stefano Bistarelli and Francesco Santini. C-semiring frameworks for minimum spanning tree problems. In *International Workshop on Algebraic Development Techniques*, pages 56–70. Springer, 2008.

[10] Sandrine Blazy, Benoît Robillard, and Andrew W. Appel. Formal verification of coalescing graph-coloring register allocation. In Andrew D. Gordon, editor, *European Symposium on Programming*, volume 6012 of *Lecture Notes in Computer Science*, pages 145–164. Springer, 2010.

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

[12] Sylvie Boldo, Catherine Lelay, and Guillaume Melquiond. Formalization of real analysis: A survey of proof assistants and libraries. *Mathematical Structures in Computer Science*, 26(7):1196–1233, 2016.

[13] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.

[14] Otakar Borůvka. O jistém problému minimálním (about a certain minimal problem). 3:33–58, 1926.

[15] Robert S. Boyer, Matt Kaufmann, and J Strother Moore. The Boyer-Moore theorem prover and its interactive enhancement. *Computers & Mathematics with Applications*, 29(2):27–62, 1995.

[16] Bernard Chazelle. A minimum spanning tree algorithm with inverse-Ackermann type complexity. *Journal of the ACM*, 47(6):1028–1047, 2000.

[17] C. C. Chen and G. Grätzer. Stone lattices. I: Construction theorems. *Canadian Journal of Mathematics*, 21:884–894, 1969.

[18] Gustave Choquet. Étude de certains réseaux de routes. *Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences*, 206:310–313, 1938.

[19] John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.

[20] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT press, 3rd edition, 2009.

[21] Brian A. Davey and Hilary A. Priestley. *Introduction to Lattices and Order*. Cambridge, second edition, 2002.

[22] Christian Doczkal and Damien Pous. Graph theory in Coq: Minors, treewidth, and isomorphisms. *Journal of Automated Reasoning*, pages 1–31, 2020.

[23] Burak Ekici, Alain Mebsout, Cesare Tinelli, Chantal Keller, Guy Katz, Andrew Reynolds, and Clark Barrett. SMTCoq: A plug-in for integrating SMT solvers into Coq. In *International Conference on Computer Aided Verification*, pages 126–133. Springer, 2017.

[24] Shimon Even. *Graph Algorithms*. Cambridge, second edition, 2012.

[25] Kazimierz Florek, Jan Łukaszewicz, Julian Perkal, Hugo Steinhaus, and Stefan Zubrzycki. Sur la liaison et la division des points d'un ensemble fini. In *Colloquium Mathematicae*, volume 2, pages 282–285, 1951.

[26] Jean-Claude Fournier. *Graph Theory and Applications*. Wiley, 2009.

[27] Michael L. Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[28] Harold N Gabow, Zvi Galil, and Thomas H. Spencer. Efficient implementation of graph algorithms using contraction. In *25th Annual Symposium on Foundations of Computer Science*, pages 347–357. IEEE, 1984.

[29] Robert G. Gallager, Pierre A. Humblet, and Philip M. Spira. A distributed algorithm for minimum-weight spanning trees. *ACM Transactions on Programming Languages and systems*, 5(1):66–77, 1983.

[30] Alan Gibbons. *Algorithmic Graph Theory*. Cambridge, 1985.

[31] Michel Gondran and Michel Minoux. *Graphs, Dioids and Semirings: New Models and Algorithms*, volume 41. Springer, 2008.

[32] Michael J. C. Gordon and Tom F. Melham. *Introduction to HOL: A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, 1993.

[33] Ronald Gould. *Graph Theory*. Benjamin/Cummings, 1988.

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

[35] George Grätzer. *General Lattice Theory*. American Press, 1978.

[36] George Grätzer. *Lattice Theory: Foundation*. Springer, 2011.

[37] George Grätzer and Elégius T Schmidt. On a problem of MH Stone. *Acta Mathematica Academiae Scientiarum Hungaricae*, 8:455–460, 1957.

[38] W. Guttmann. Relation-algebraic verification of Prim's minimum spanning tree algorithm. In A. Sampaio and F. Wang, editors, *Theoretical Aspects of Computing*, volume 9965 of *Lecture Notes in Computer Science*, pages 51–68. Springer, 2016.

[39] W. Guttmann. Stone-Kleene relation algebras. *Archive of Formal Proofs*, 2017.

[40] W. Guttmann. Stone relation algebras. *Archive of Formal Proofs*, 2017.

[41] W. Guttmann. Aggregation algebras. *Archive of Formal Proofs*, 2018.

[42] Walter Guttmann. Stone relation algebras. In Peter Höfner, Damien Pous, and Georg Struth, editors, *International Conference on Relational and Algebraic Methods in Computer Science*, volume 10226 of *Lecture Notes in Computer Science*, pages 127–143. Springer, 2017.

[43] Walter Guttmann. An algebraic framework for minimum spanning tree problems. *Theoretical Computer Science*, 744:37–55, 2018.

[44] Walter Guttmann. Verifying minimum spanning tree algorithms with Stone relation algebras. *Journal of Logical and Algebraic Methods in Programming*, 101:132–150, 2018.

[45] Walter Guttmann. Verifying the correctness of disjoint-set forests with Kleene relation algebras. In Uli Fahrenberg, Peter Jipsen, and Michael Winter, editors, *Relational and Algebraic Methods in Computer Science*, volume 12062 of *Lecture Notes in Computer Science*, pages 134–151. Springer, 2020.

[46] Norah Hartsfield and Gerhard Ringel. *Pearls in Graph Theory: A Comprehensive Introduction*. Academic Press, 1994.

[47] Egbert Harzheim. *Ordered Sets*, volume 7 of *Advances in Mathematics*. Springer, 2005.

[48] Wim H. Hesselink. The verified incremental design of a distributed spanning tree algorithm. *Formal Aspects of Computing*, 11(1):45–55, 1999.

[49] Peter Höfner and Bernhard Möller. Dijkstra, Floyd and Warshall meet Kleene. *Formal Aspects of Computing*, 24(4-6):459–476, 2012.

[50] Thomas Judson. *Abstract Algebra: Theory and Applications*. Austin State University, 2016.

[51] David R Karger, Philip N Klein, and Robert E Tarjan. A randomized linear-time algorithm to find minimum spanning trees. *Journal of the ACM*, 42(2):321–328, 1995.

[52] Britta Kehden and Frank Neumann. A relation-algebraic view on evolutionary algorithms for some graph problems. In Jens Gottlieb and Günther R. Raidl, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 3906 of *Lecture Notes in Computer Science*, pages 147–158. Springer, 2006.

[53] Dexter Kozen. A completeness theorem for Kleene algebras and the algebra of regular events. *Information and Computation*, 110(2):366–390, 1994.

[54] Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956.

[55] Peter Lammich and S. Reza Sefidgar. Formalizing network flow algorithms: A refinement approach in Isabelle/HOL. *Journal of Automated Reasoning*, 62(2):261–280, 2019.

[56] Gilbert Lee. Verification of graph algorithms in Mizar. Master's thesis, University of Alberta, 2004.

[57] Roger D. Maddux. The origin of relation algebras in the development and axiomatization of the calculus of relations. *Studia Logica*, 50(3-4):421–455, 1991.

[58] Roger D. Maddux. Relation-algebraic semantics. *Theoretical Computer Science*, 160(1-2):1–85, 1996.

[59] Roger D. Maddux. Relation algebras. In *Studies in Logic and the Foundation of Mathematics*, volume 150. Elsevier, 2006.

[60] Martin Mareš. The saga of minimum spanning trees. *Computer Science Review*, 2(3):165–221, 2008.

[61] Mehryar Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[62] Michal Muzalewski. *An Outline of PC Mizar*. Fondation Philippe le Hodey, 1993.

[63] Jaroslav Nešetřil, Eva Milková, and Helena Nešetřilová. Otakar Borůvka on minimum spanning tree problem translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1-3):3–36, 2001.

[64] Jaroslav Nešetril and Helena Nešetrilová. The origins of minimal spanning tree algorithms–Boruvka and Jarník. *Documenta Mathematica, Extra Volume on Optimization Stories*, pages 127–141, 2012.

[65] Tobias Nipkow. Winskel is (almost) right: Towards a mechanized semantics textbook. *Formal Aspects of Computing*, 10(2):171–186, 1998.

[66] Tobias Nipkow. Hoare logics in Isabelle/HOL. In Helmut Schwichtenberg and Ralf Steinbrüggen, editors, *Proof and System-Reliability*, pages 341–367. Kluwer Academic Publishers, 2002.

[67] Tobias Nipkow, Gertrud Bauer, and Paula Schultz. Flyspeck I: tame graphs. In *International Joint Conference on Automated Reasoning*, pages 21–35. Springer, 2006.

[68] Tobias Nipkow, Manuel Eberl, and Maximilian P. L. Haslbeck. Verified textbook algorithms. A biased survey. In Dang Van Hung and Oleg Sokolsky, editors, *ATVA 2020, Automated Technology for Verification and Analysis*, Lecture Notes in Computer Science. Springer, 2020. To appear.

[69] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, 2002.

[70] Lars Noschinski. A graph library for Isabelle. *Mathematics in Computer Science*, 9(1):23–39, 2015.

[71] Eugenio G. Omodeo, Domenico Cantone, Alberto Policriti, and Jacob T. Schwartz. A computerized referee. In *Reasoning, Action and Interaction in AI Theories and Systems*, pages 117–139. Springer, 2006.

[72] Eugenio G. Omodeo and Alexandru I. Tomescu. Set graphs. III. Proof pearl: Claw-free graphs mirrored into transitive hereditarily finite sets. *Journal of Automated Reasoning*, 52(1):1–29, 2014.

[73] J. C. Paulson, L. C. Blanchette. Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. In Ternovska E. Sutcliffe G., Schulz S., editor, *International Workshop on the Implementation of Logics*, 2010.

[74] Robert Clay Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957.

[75] Saidur Rahman. *Basic Graph Theory*. Springer, 2017.

[76] Steven Roman. *Lattices and Ordered Sets*. Springer, 2008.

[77] Gunther Schmidt and Thomas Ströhlein. *Relations and Graphs–Discrete Mathematics for Computer Scientists*. EATCS Monographs on Theoretical Computer Science. Springer, 1993.

[78] Steven S. Skiena. *The Algorithm Design Manual*, volume 1. Springer, second edition, 1998.

[79] M. Sollin. La trace de canalisation. In C. Berge and A. Ghouilla-Houri, editors, *Programming, Games, and Transportation Networks*. Wiley, 1965.

[80] Robert Endre Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics*. SIAM, 1983.

[81] Alfred Tarski. On the calculus of relations. *The Journal of Symbolic Logic*, 6(3):73–89, 1941.

[82] Douglas Brent West. *Introduction to Graph Theory*. Pearson, second edition, 2002.

[83] Freek Wiedijk. *The Seventeen Provers of the World: Foreword by Dana S. Scott*, volume 3600 of *Lecture Notes in Artificial Intelligence*. Springer, 2006.

[84] Andrew Chi-Chih Yao. An $O(e \log \log v)$ algorithm for finding minimum spanning trees. *Information Processing Letters*, 4(1):21–23, 1975.

Appendix A

An intuition for the weighted-graph instance notation

We denote the set of matrices whose entries range over the real numbers, extended by $\top$ and $\bot$ as $\mathbb{R}'^{A\times A}$. Here, we give some intuition for this.

The set of extended real numbers, $\mathbb{R}'$, is the union of the real numbers with the set $\{\bot, \top\}$, that is $\mathbb{R}' = \mathbb{R} \cup \{\bot, \top\}$. A notation that is sometimes used to denote the set of functions from set X to set Y is Y^X. Before discussing how this applies for $\mathbb{R}'^{A\times A}$ we give a simple example.

Note that 2^S denotes the set of functions mapping elements from the set S to the two-element set. A common use of this particular notation is to denote the power-set relation (function). This is because we can interpret one element of the two-element set to denote inclusion (for instance 1) and the other to denote exclusion (for instance 0).

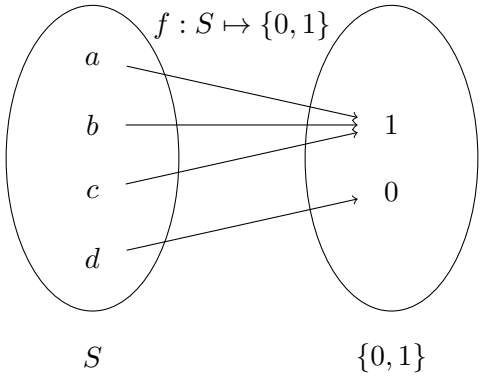
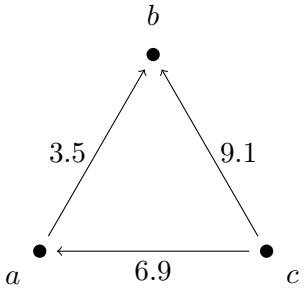


Figure A.1: A depiction of how 2^S denotes a power set. Here, f maps S to $\{0, 1\}$ in a way that denotes the subset $\{a, b, c\}$.

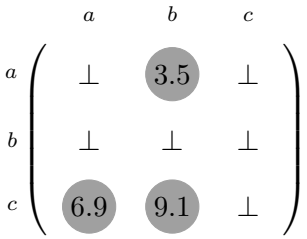
Consider the set $S = \{a, b, c, d\}$ and $\{0, 1\}$ in Figure A.1. The instance of f shown maps all elements of S to 1 except for d which is mapped to 0. This is the instance of 2^S that denotes the subset, $\{a, b, c\}$. The function mapping all elements of S to 0 would denote the empty set.

Next we consider $\mathbb{R}'^{A\times A}$. Our interpretation of this is the set of functions mapping the set of tuples of the Cartesian product, $A \times A$, to the extended real numbers. Recall, that A denotes the index set of vertices under consideration. Therefore, $\mathbb{R}'^{A\times A}$ denotes all possible weighted-graph instances, as matrices, that may be formed over graphs with a vertex set, A, and edge weights taken from $\mathbb{R}'$.

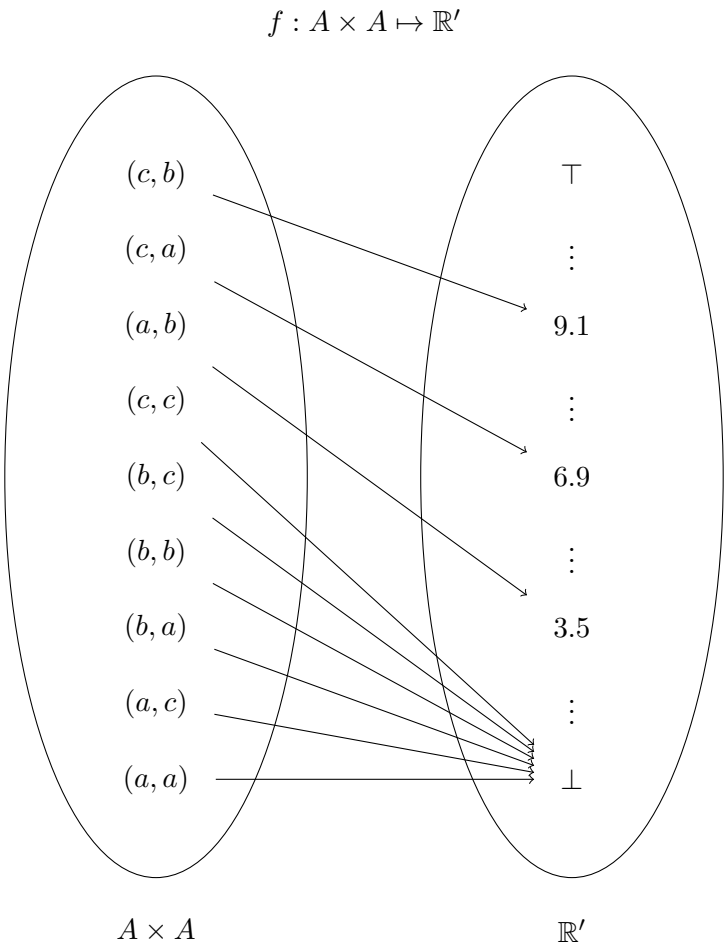
Consider, for example, the graph in Figure A.2(a) that has three edges with weights: 3.5, 9.1, and 6.9. This graph is shown in matrix form in Figure A.2(b). In Figure A.2(c), the function, $f : A \times A \mapsto \mathbb{R}'$, maps vertex pairs to edge weights for the graph. For example, we see that (c, a) from set $A \times A$ maps to 6.9 from set $\mathbb{R}'$. Likewise (a, a) maps to $\bot$.



(a) A graph of three vertices, a, b, and c. Edge weights are selected from the extended real numbers.



(b) A matrix view of the graph from Figure A.2(a).



(c) A particular function, f, from the set of functions $\mathbb{R}'^{A\times A}$ that denote possible matrices over index set A. Here, f represents the weighted-graph instance from Figure A.2(a).

Figure A.2: A depiction of how $\mathbb{R}'^{A\times A}$ denotes all possible weighted-graph instances, as matrices, that may be formed over graphs with a vertex set, A, and edge weights taken from $\mathbb{R}'$.

Appendix B

Isabelle/HOL theory

We include the Isabelle/HOL theory that formally verifies the correctness of Borůvka's MST algorithm in this appendix. We intend to publish this theory to the Archive of Formal Proofs. We also include results about weakly connected components [45] that have not yet been published to the Archive of Formal Proofs. Until these theories are published to the Archive of Formal Proofs, they will also be available at https://gitlab.com/nicobrien/boruvka-mst-theory.

B.1 Weakly connected components

The results in this section are from [45].

theory *WCC*

imports *Stone-Kleene-Relation-Algebras.Kleene-Relation-Algebras*

begin

no-notation
trancl ((-$^+$) [*1000*] *999*)

context *stone-kleene-relation-algebra*
begin

lemma *reachable-without-loops*:
$x^\star = (x \sqcap -1)^\star$
proof (*rule antisym*)
have $x * (x \sqcap -1)^\star = (x \sqcap 1) * (x \sqcap -1)^\star \sqcup (x \sqcap -1) * (x \sqcap -1)^\star$
by (*metis maddux-3-11-pp mult-right-dist-sup regular-one-closed*)
also have ... $\leq (x \sqcap -1)^\star$
by (*metis inf.cobounded2 le-supI mult-left-isotone star.circ-circ-mult star.left-plus-below-circ star-involutive star-one*)
finally show $x^\star \leq (x \sqcap -1)^\star$
by (*metis inf.cobounded2 maddux-3-11-pp regular-one-closed star.circ-circ-mult star.circ-sup-2 star-involutive star-sub-one*)
next
show $(x \sqcap -1)^\star \leq x^\star$
by (*simp add*: *star-isotone*)
qed

abbreviation *wcc x* $\equiv (x \sqcup x^T)^\star$

lemma *wcc-equivalence*:
equivalence (*wcc x*)
apply (*intro conjI*)

subgoal by (*simp add*: *star.circ-reflexive*)
subgoal by (*simp add*: *star.circ-transitive-equal*)
subgoal by (*simp add*: *conv-dist-sup conv-star-commute sup-commute*)
done

lemma *wcc-increasing*:
$x \leq wcc\ x$
by (*simp add*: *star.circ-sub-dist-1*)

lemma *wcc-isotone*:
$x \leq y \Longrightarrow wcc\ x \leq wcc\ y$
using *conv-isotone star-isotone sup-mono* **by** *blast*

lemma *wcc-idempotent*:
$wcc\ (wcc\ x) = wcc\ x$
using *star-involutive wcc-equivalence* **by** *auto*

lemma *wcc-below-wcc*:
$x \leq wcc\ y \Longrightarrow wcc\ x \leq wcc\ y$
using *wcc-idempotent wcc-isotone* **by** *fastforce*

lemma *wcc-bot*:
$wcc\ bot = 1$
by (*simp add*: *star.circ-zero*)

lemma *wcc-one*:
$wcc\ 1 = 1$
by (*simp add*: *star-one*)

lemma *wcc-top*:
$wcc\ top = top$
by (*simp add*: *star.circ-top*)

lemma *wcc-with-loops*:
$wcc\ x = wcc\ (x \sqcup 1)$
using *conv-dist-sup star-decompose-1 star-sup-one sup-commute symmetric-one-closed* **by** *presburger*

lemma *wcc-without-loops*:
$wcc\ x = wcc\ (x \sqcap -1)$
by (*metis conv-star-commute star-sum reachable-without-loops*)

lemma *forest-components-wcc*:
$injective\ x \Longrightarrow wcc\ x = forest\text{-}components\ x$
by (*simp add*: *cancel-separate-1*)

abbreviation $fc\ x \equiv x^\star * x^{T\star}$

lemma *fc-equivalence*:
$univalent\ x \Longrightarrow equivalence\ (fc\ x)$
apply (*intro conjI*)
subgoal by (*simp add*: *reflexive-mult-closed star.circ-reflexive*)
subgoal by (*metis cancel-separate-1 eq-iff star.circ-transitive-equal*)
subgoal by (*simp add*: *conv-dist-comp conv-star-commute*)
done

lemma *fc-increasing*:
$x \leq fc\ x$
by (*metis le-supE mult-left-isotone star.circ-back-loop-fixpoint star.circ-increasing*)

lemma *fc-isotone*:
$x \leq y \Longrightarrow fc\ x \leq fc\ y$
by (*simp add*: *comp-isotone conv-isotone star-isotone*)

lemma *fc-idempotent*:
univalent $x \Longrightarrow fc\ (fc\ x) = fc\ x$
by (*metis fc-equivalence cancel-separate-1 star.circ-transitive-equal star-involutive*)

lemma *fc-star*:
univalent $x \Longrightarrow (fc\ x)^\star = fc\ x$
using *fc-equivalence fc-idempotent star.circ-transitive-equal* **by** *simp*

lemma *fc-plus*:
univalent $x \Longrightarrow (fc\ x)^+ = fc\ x$
by (*metis fc-star star.circ-decompose-9*)

lemma *fc-bot*:
$fc\ bot = 1$
by (*simp add*: *star.circ-zero*)

lemma *fc-one*:
$fc\ 1 = 1$
by (*simp add*: *star-one*)

lemma *fc-top*:
$fc\ top = top$
by (*simp add*: *star.circ-top*)

lemma *fc-wcc*:
univalent $x \Longrightarrow wcc\ x = fc\ x$
by (*simp add*: *fc-star star-decompose-1*)

end

end

B.2 Borůvka's minimum spanning tree algorithm

In this section we prove partial-correctness of Borůvka's minimum spanning tree algorithm. The specification is similar as for Guttmann's proof of Kruskal's minimum spanning tree algorithm and the proof is conducted in Stone-Kleene relation algebras supplemented with the Tarski rule for regular elements, operations for aggregation and minimization, and an operation to select components of a graph. The proof uses Hoare Logic.

theory *Boruvka*

imports
Aggregation-Algebras.Minimum-Spanning-Trees
WCC

begin

B.2.1 General results

The proof of Borůvka's minimum spanning tree algorithm is carried out in *m-k*-Stone-Kleene relation algebras, that is, *stone-kleene-relation-algebra-tarski*. In this section we give results that

hold more generally.

context *stone-kleene-relation-algebra*
begin

definition *big-forest H d* ≡
  equivalence H
∧ $d \leq -H$
∧ *univalent* $(H * d)$
∧ $H \sqcap d * d^T \leq 1$
∧ $(H * d)^+ \leq - H$

definition *bf-between-points p q H d* ≡ *point p* ∧ *point q* ∧ $p \leq (H * d)^\star * H * d$
definition *bf-between-arcs a b H d* ≡ *arc a* ∧ *arc b* ∧ $a^T * top \leq (H * d)^\star * H * b * top$
definition *e-forest-path a b H d g* ≡
  big-forest H d
∧ *arc a*
∧ *arc b*
∧ $a^T * top \leq (H * d)^\star * H * b * top$
∧ $a \leq -H \sqcap --g$
∧ $b \leq d$

Theorem 3

lemma *He-eq-He-THe-star*:
  assumes *arc e*
    and *equivalence H*
  shows $H * e = H * e * (top * H * e)^\star$
proof −
  let *?x* = $H * e$
  have *1*: $H * e \leq H * e * (top * H * e)^\star$
    using *comp-isotone star.circ-reflexive* **by** *fastforce*
  have $H * e * (top * H * e)^\star = H * e * (top * e)^\star$
    by (*metis assms*(*2*) *preorder-idempotent surjective-var*)
  also have ... $\leq H * e * (1 \sqcup top * (e * top)^\star * e)$
    by (*metis eq-refl star.circ-mult-1*)
  also have ... $\leq H * e * (1 \sqcup top * top * e)$
    by (*simp add*: *star.circ-left-top*)
  also have ... $= H * e \sqcup H * e * top * e$
    by (*simp add*: *mult.semigroup-axioms semiring.distrib-left semigroup.assoc*)
  finally have *2*: $H * e * (top * H * e)^\star \leq H * e$
    using *assms arc-top-arc mult-assoc* **by** *auto*
  thus *?thesis*
    using *1 2* **by** *simp*
qed

lemma *path-through-components*:
  assumes *equivalence H*
    and *arc e*
  shows $(H * (d \sqcup e))^\star = (H * d)^\star \sqcup (H * d)^\star * H * e * (H * d)^\star$
proof −
  have $H * e * (H * d)^\star * H * e \leq H * e * top * H * e$
    by (*simp add*: *comp-isotone*)
  also have ... $= H * e * top * e$
    by (*metis assms*(*1*) *preorder-idempotent surjective-var mult-assoc*)
  also have ... $= H * e$
    using *assms*(*2*) *arc-top-arc mult-assoc* **by** *auto*
  finally have *1*: $H * e * (H * d)^\star * H * e \leq H * e$
    by *simp*
  have $(H * (d \sqcup e))^\star = (H * d \sqcup H * e)^\star$

by (*simp add*: *comp-left-dist-sup*)
also have ... = $(H * d)^\star \sqcup (H * d)^\star * H * e * (H * d)^\star$
using *1 star-separate-3* **by** (*simp add*: *mult-assoc*)
finally show *?thesis*
by *simp*
qed

lemma *simplify-f*:
assumes *regular p*
and *regular e*
shows $(f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap - e^T \sqcap p) \sqcup (f \sqcap - e^T \sqcap p)^T \sqcup (f \sqcap - e^T \sqcap - p)^T \sqcup e^T \sqcup e = f \sqcup f^T \sqcup e \sqcup e^T$
proof −
have $(f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap - e^T \sqcap p) \sqcup (f \sqcap - e^T \sqcap p)^T \sqcup (f \sqcap - e^T \sqcap - p)^T \sqcup e^T \sqcup e$
$= (f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap - e^T \sqcap p) \sqcup (f^T \sqcap - e \sqcap p^T) \sqcup (f^T \sqcap - e \sqcap - p^T) \sqcup e^T \sqcup e$
by (*simp add*: *conv-complement conv-dist-inf*)
also have ... =
$((f \sqcup (f \sqcap - e^T \sqcap p)) \sqcap (- e^T \sqcup (f \sqcap - e^T \sqcap p)) \sqcap (- p \sqcup (f \sqcap - e^T \sqcap p)))$
$\sqcup ((f^T \sqcup (f^T \sqcap - e \sqcap - p^T)) \sqcap (- e \sqcup (f^T \sqcap - e \sqcap - p^T)) \sqcap (p^T \sqcup (f^T \sqcap - e \sqcap - p^T)))$
$\sqcup e^T \sqcup e$
using *sup-inf-distrib2 sup-assoc* **by** *presburger*
also have ... =
$((f \sqcup f) \sqcap (f \sqcup - e^T) \sqcap (f \sqcup p) \sqcap (- e^T \sqcup f) \sqcap (- e^T \sqcup - e^T) \sqcap (- e^T \sqcup p) \sqcap (- p \sqcup f) \sqcap (- p \sqcup - e^T) \sqcap (- p \sqcup p))$
$\sqcup ((f^T \sqcup f^T) \sqcap (f^T \sqcup - e) \sqcap (f^T \sqcup - p^T) \sqcap (- e \sqcup f^T) \sqcap (- e \sqcup - e) \sqcap (- e \sqcup - p^T) \sqcap (p^T \sqcup f^T) \sqcap (p^T \sqcup - e) \sqcap (p^T \sqcup - p^T))$
$\sqcup e^T \sqcup e$
using *sup-inf-distrib1 sup-assoc inf-assoc sup-inf-distrib1* **by** *simp*
also have ... =
$((f \sqcup f) \sqcap (f \sqcup - e^T) \sqcap (f \sqcup p) \sqcap (f \sqcup - p) \sqcap (- e^T \sqcup f) \sqcap (- e^T \sqcup - e^T) \sqcap (- e^T \sqcup p) \sqcap (- e^T \sqcup - p) \sqcap (- p \sqcup p))$
$\sqcup ((f^T \sqcup f^T) \sqcap (f^T \sqcup - e) \sqcap (f^T \sqcup - p^T) \sqcap (- e \sqcup f^T) \sqcap (f^T \sqcup p^T) \sqcap (- e \sqcup - e) \sqcap (- e \sqcup - p^T) \sqcap (- e \sqcup p^T) \sqcap (p^T \sqcup - p^T))$
$\sqcup e^T \sqcup e$
by (*smt abel-semigroup.commute inf.abel-semigroup-axioms inf.left-commute sup.abel-semigroup-axioms*)
also have ... = $(f \sqcap - e^T \sqcap (- p \sqcup p)) \sqcup (f^T \sqcap - e \sqcap (p^T \sqcup - p^T)) \sqcup e^T \sqcup e$
by (*smt inf.sup-monoid.add-assoc inf.sup-monoid.add-commute inf-sup-absorb sup.idem*)
also have ... = $(f \sqcap - e^T) \sqcup (f^T \sqcap - e) \sqcup e^T \sqcup e$
by (*metis assms(1) conv-complement inf-top-right stone*)
also have ... = $(f \sqcup e^T) \sqcap (- e^T \sqcup e^T) \sqcup (f^T \sqcup e) \sqcap (- e \sqcup e)$
by (*metis sup.left-commute sup-assoc sup-inf-distrib2*)
finally show *?thesis*
by (*metis abel-semigroup.commute assms(2) conv-complement inf-top-right stone sup.abel-semigroup-axioms sup-assoc*)
qed

lemma *simplify-forest-components-f*:
assumes *regular p*
and *regular e*
and *injective* $(f \sqcap - e^T \sqcap - p \sqcup (f \sqcap - e^T \sqcap p)^T \sqcup e)$
and *injective f*
shows *forest-components* $((f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap -e^T \sqcap p)^T \sqcup e) = (f \sqcup f^T \sqcup e \sqcup e^T)^\star$
proof −
have *forest-components* $((f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap -e^T \sqcap p)^T \sqcup e) = wcc\ ((f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap - e^T \sqcap p)^T \sqcup e)$
by (*simp add*: *assms(3) forest-components-wcc*)
also have ... = $((f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap - e^T \sqcap p)^T \sqcup e \sqcup (f \sqcap - e^T \sqcap - p)^T \sqcup (f \sqcap - e^T \sqcap p)$

$\sqcup\ e^T)^\star$
using *conv-dist-sup sup-assoc* **by** *auto*
also have ... = $((f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap - e^T \sqcap p) \sqcup (f \sqcap - e^T \sqcap p)^T \sqcup (f \sqcap - e^T \sqcap - p)^T \sqcup e^T \sqcup e)^\star$
using *sup-assoc sup-commute* **by** *auto*
also have ... = $(f \sqcup f^T \sqcup e \sqcup e^T)^\star$
using *assms*(*1*, *2*, *3*, *4*) *simplify-f* **by** *auto*
finally show *?thesis*
by *simp*
qed

lemma *components-disj-increasing*:
assumes *regular p*
and *regular e*
and *injective* $(f \sqcap - e^T \sqcap - p \sqcup (f \sqcap - e^T \sqcap p)^T \sqcup e)$
and *injective f*
shows *forest-components f* $\leq$ *forest-components* $(f \sqcap - e^T \sqcap - p \sqcup (f \sqcap - e^T \sqcap p)^T \sqcup e)$
proof −
have *1*: *forest-components* $((f \sqcap - e^T \sqcap - p) \sqcup (f \sqcap -e^T \sqcap p)^T \sqcup e) = (f \sqcup f^T \sqcup e \sqcup e^T)^\star$
using *simplify-forest-components-f assms*(*1*, *2*, *3*, *4*) **by** *blast*
have *forest-components f* = *wcc f*
by (*simp add*: *assms*(*4*) *forest-components-wcc*)
also have ... $\leq (f \sqcup f^T \sqcup e^T \sqcup e)^\star$
by (*simp add*: *le-supI2 star-isotone sup-commute*)
finally show *?thesis*
using *1 sup.left-commute sup-commute* **by** *simp*
qed

lemma *fch-equivalence*:
assumes *forest h*
shows *equivalence* (*forest-components h*)
by (*simp add*: *assms*(*1*) *forest-components-equivalence*)

lemma *big-forest-path-split-1*:
assumes *arc a*
and *equivalence H*
shows $(H * d)^\star * H * a * top = (H * (d \sqcap - a))^\star * H * a * top$
proof −
let *?H* = *H*
let *?x* = *?H* $*$ $(d \sqcap -a)$
let *?y* = *?H* $*$ *a*
let *?a* = *?H* $*$ *a* $*$ *top*
let *?d* = *?H* $*$ *d*
have *1*: $?d^\star * ?a \leq ?x^\star * ?a$
proof −
have $?x^\star * ?y * ?x^\star * ?a \leq ?x^\star * ?a * ?a$
using *mult-left-isotone star.circ-right-top top-right-mult-increasing mult-assoc* **by** *smt*
also have ... = $?x^\star * ?a * a * top$
by (*metis ex231e mult-assoc*)
also have ... = $?x^\star * ?a$
by (*simp add*: *assms*(*1*) *mult-assoc*)
finally have *11*: $?x^\star * ?y * ?x^\star * ?a \leq ?x^\star * ?a$
by *simp*
have $?d^\star * ?a = (?H * (d \sqcap a) \sqcup ?H * (d \sqcap -a))^\star * ?a$
proof −
have *12*: *regular a*
using *assms*(*1*) *arc-regular* **by** *simp*
have $?H * ((d \sqcap a) \sqcup (d \sqcap - a)) = ?H * (d \sqcap top)$

```
      using 12 by (metis inf-top-right maddux-3-11-pp)
    thus ?thesis
      using mult-left-dist-sup by auto
  qed
  also have ... ≤ (?y ⊔ ?x)^⋆ * ?a
    by (metis comp-inf.coreflexive-idempotent comp-isotone inf.cobounded1
inf.sup-monoid.add-commute semiring.add-mono star-isotone top.extremum)
  also have ... = (?x ⊔ ?y)^⋆ * ?a
    by (simp add: sup-commute mult-assoc)
  also have ... = ?x^⋆ * ?a ⊔ (?x^⋆ * ?y * (?x^⋆ * ?y)^⋆ * ?x^⋆) * ?a
    by (smt mult-right-dist-sup star.circ-sup-9 star.circ-unfold-sum mult-assoc)
  also have ... ≤ ?x^⋆ * ?a ⊔ (?x^⋆ * ?y * (top * ?y)^⋆ * ?x^⋆) * ?a
  proof -
    have (?x^⋆ * ?y)^⋆ ≤ (top * ?y)^⋆
      by (simp add: mult-left-isotone star-isotone)
    thus ?thesis
      by (metis comp-inf.coreflexive-idempotent comp-inf.transitive-star eq-refl mult-left-dist-sup
top.extremum mult-assoc)
  qed
  also have ... = ?x^⋆ * ?a ⊔ (?x^⋆ * ?y * ?x^⋆) * ?a
    using assms(1, 2) He-eq-He-THe-star arc-regular mult-assoc by auto
  finally have 13: (?H * d)^⋆ * ?a ≤ ?x^⋆ * ?a ⊔ ?x^⋆ * ?y * ?x^⋆ * ?a
    by (simp add: mult-assoc)
  have 14: ?x^⋆ * ?y * ?x^⋆ * ?a ≤ ?x^⋆ * ?a
    using 11 mult-assoc by auto
  thus ?thesis
    using 13 14 sup.absorb1 by auto
qed
have 2: ?d^⋆ * ?a ≥ ?x^⋆ *?a
  by (simp add: comp-isotone star-isotone)
thus ?thesis
  using 1 2 antisym mult-assoc by simp
qed

lemma dTransHd-le-1:
assumes equivalence H
  and univalent (H * d)
shows d^T * H * d ≤ 1
proof -
have d^T * H^T * H * d ≤ 1
  using assms(2) conv-dist-comp mult-assoc by auto
thus ?thesis
  using assms(1) mult-assoc by (simp add: preorder-idempotent)
qed

lemma HcompaT-le-compHaT:
assumes equivalence H
  and injective (a * top)
shows -H * a * top ≤ - (H * a * top)
proof -
have a * top * a^T ≤ 1
  by (metis assms(2) conv-dist-comp symmetric-top-closed vector-top-closed mult-assoc)
then have a * top * a^T * H ≤ H
  using assms(1) comp-isotone order-trans by blast
then have a * top * top * a^T * H ≤ H
  by (simp add: vector-mult-closed)
then have a * top * (H * a * top)^T ≤ H
  by (metis assms(1) conv-dist-comp symmetric-top-closed vector-top-closed mult-assoc)
```

thus *?thesis*
using *assms(2) comp-injective-below-complement mult-assoc* **by** *auto*
qed

Theorem 4

lemma *expand-big-forest*:
assumes *big-forest H d*
shows $(d^T * H)^\star * (H * d)^\star = (d^T * H)^\star \sqcup (H * d)^\star$
proof −
have $(H * d)^T * H * d \leq 1$
using *assms big-forest-def mult-assoc* **by** *auto*
then have $d^T * H * H * d \leq 1$
using *assms big-forest-def conv-dist-comp* **by** *auto*
thus *?thesis*
by (*simp add*: *cancel-separate-eq comp-associative*)
qed

lemma *big-forest-path-bot*:
assumes *arc a*
and $a \leq d$
and *big-forest H d*
shows $(d \sqcap - a)^T * (H * a * top) \leq bot$
proof −
have *1*: $d^T * H * d \leq 1$
using *assms(3) big-forest-def dTransHd-le-1* **by** *blast*
then have $d * - 1 * d^T \leq - H$
using *triple-schroeder-p* **by** *force*
then have $d * - 1 * d^T \leq 1 \sqcup - H$
by (*simp add*: *le-supI2*)
then have $d * d^T \sqcup d * - 1 * d^T \leq 1 \sqcup - H$
using *assms(3) big-forest-def inf-commute regular-one-closed shunting-p* **by** (*metis le-supI*)
then have $d * 1 * d^T \sqcup d * - 1 * d^T \leq 1 \sqcup - H$
by *simp*
then have $d * (1 \sqcup - 1) * d^T \leq 1 \sqcup - H$
using *comp-associative mult-right-dist-sup* **by** (*simp add*: *mult-left-dist-sup*)
then have $d * top * d^T \leq 1 \sqcup - H$
using *regular-complement-top* **by** *auto*
then have $d * top * a^T \leq 1 \sqcup - H$
using *assms(2) conv-isotone dual-order.trans mult-right-isotone* **by** *blast*
then have $d * (a * top)^T \leq 1 \sqcup - H$
by (*simp add*: *comp-associative conv-dist-comp*)
then have $d \leq (1 \sqcup - H) * (a * top)$
by (*simp add*: *assms(1) shunt-bijective*)
then have $d \leq a * top \sqcup - H * a * top$
by (*simp add*: *comp-associative mult-right-dist-sup*)
also have ... $\leq a * top \sqcup - (H * a * top)$
using *assms(1, 3) HcompaT-le-compHaT big-forest-def sup-right-isotone* **by** *auto*
finally have $d \leq a * top \sqcup - (H * a * top)$
by *simp*
then have $d \sqcap --(H * a * top) \leq a * top$
using *shunting-var-p* **by** *auto*
then have *2*:$d \sqcap H * a * top \leq a * top$
using *inf.sup-right-isotone order.trans pp-increasing* **by** *blast*
have *3*:$d \sqcap H * a * top \leq top * a$
proof −
have $d \sqcap H * a * top \leq (H * a \sqcap d * top^T) * (top \sqcap (H * a)^T * d)$
by (*metis dedekind inf-commute*)

```
    also have ... = d * top ⊓ H * a * a^T * H^T * d
      by (simp add: conv-dist-comp inf-vector-comp mult-assoc)
    also have ... ≤ d * top ⊓ H * a * d^T * H^T * d
      using assms(2) mult-right-isotone mult-left-isotone conv-isotone inf.sup-right-isotone by auto
    also have ... = d * top ⊓ H * a * d^T * H * d
      using assms(3) big-forest-def by auto
    also have ... ≤ d * top ⊓ H * a * 1
      using 1 by (metis inf.sup-right-isotone mult-right-isotone mult-assoc)
    also have ... ≤ H * a
      by simp
    also have ... ≤ top * a
      by (simp add: mult-left-isotone)
    finally have d ⊓ H * a * top ≤ top * a
      by simp
    thus ?thesis
      by simp
  qed
  have d ⊓ H * a * top ≤ a * top ⊓ top * a
    using 2 3 by simp
  also have ... = a * top * top * a
    by (metis comp-associative comp-inf.star.circ-decompose-9 comp-inf.star-star-absorb
comp-inf-covector vector-inf-comp vector-top-closed)
  also have ... = a * top * a
    by (simp add: vector-mult-closed)
  finally have 4:d ⊓ H * a * top ≤ a
    by (simp add: assms(1) arc-top-arc)
  then have d ⊓ - a ≤ -(H * a * top)
    using assms(1) arc-regular p-shunting-swap by fastforce
  then have (d ⊓ - a) * top ≤ -(H * a * top)
    using mult.semigroup-axioms p-antitone-iff schroeder-4-p semigroup.assoc by fastforce
  thus ?thesis
    by (simp add: schroeder-3-p)
qed

lemma big-forest-path-split-2:
  assumes arc a
    and a ≤ d
    and big-forest H d
  shows (H * (d ⊓ - a))^⋆ * H * a * top = (H * ((d ⊓ - a) ⊔ (d ⊓ - a)^T))^⋆ * H * a * top
proof -
  let ?lhs = (H * (d ⊓ - a))^⋆ * H * a * top
  have 1: d^T * H * d ≤ 1
    using assms(3) big-forest-def dTransHd-le-1 by blast
  have 2: H * a * top ≤ ?lhs
    by (metis le-iff-sup star.circ-loop-fixpoint star.circ-transitive-equal star-involutive sup-commute
mult-assoc)
  have (d ⊓ - a)^T * (H * (d ⊓ - a))^⋆ * (H * a * top) = (d ⊓ - a)^T * H * (d ⊓ - a) * (H * (d ⊓
- a))^⋆ * (H * a * top)
  proof -
    have (d ⊓ - a)^T * (H * (d ⊓ - a))^⋆ * (H * a * top) = (d ⊓ - a)^T * (1 ⊔ H * (d ⊓ - a) * (H *
(d ⊓ - a))^⋆) * (H * a * top)
      by (simp add: star-left-unfold-equal)
    also have ... = (d ⊓ - a)^T * H * a * top ⊔ (d ⊓ - a)^T * H * (d ⊓ - a) * (H * (d ⊓ - a))^⋆ * (H
* a * top)
      by (smt mult-left-dist-sup star.circ-loop-fixpoint star.circ-mult-1 star-slide sup-commute
mult-assoc)
    also have ... = bot ⊔ (d ⊓ - a)^T * H * (d ⊓ - a) * (H * (d ⊓ - a))^⋆ * (H * a * top)
      using assms(1, 2, 3) big-forest-path-bot mult-assoc le-bot by metis
```

```
    thus ?thesis
      by (simp add: calculation)
  qed
  also have ... ≤ d^T * H * d * (H * (d ⊓ - a))^⋆ * (H * a * top)
    using conv-isotone inf.cobounded1 mult-isotone by auto
  also have ... ≤ 1 * (H * (d ⊓ - a))^⋆ * (H * a * top)
    using 1 by (metis le-iff-sup mult-right-dist-sup)
  finally have 3: (d ⊓ - a)^T * (H * (d ⊓ - a))^⋆ * (H * a * top) ≤ ?lhs
    using mult-assoc by auto
  then have 4: H * (d ⊓ - a)^T * (H * (d ⊓ - a))^⋆ * (H * a * top) ≤ ?lhs
  proof -
    have H * (d ⊓ - a)^T * (H * (d ⊓ - a))^⋆ * (H * a * top) ≤ H * (H * (d ⊓ - a))^⋆ * H * a * top
      using 3 mult-right-isotone mult-assoc by auto
    also have ... = H * H * ((d ⊓ - a) * H)^⋆ * H * a * top
      using assms(3) big-forest-def star-slide mult-assoc preorder-idempotent by metis
    also have ... = H * ((d ⊓ - a) * H)^⋆ * H * a * top
      using assms(3) big-forest-def preorder-idempotent by fastforce
    finally show ?thesis
      by (metis assms(3) big-forest-def preorder-idempotent star-slide mult-assoc)
  qed
  have 5: (H * (d ⊓ - a) ⊔ H * (d ⊓ - a)^T) * (H * (d ⊓ - a))^⋆  * H * a * top ≤ ?lhs
  proof -
    have 51: H * (d ⊓ - a) * (H * (d ⊓ - a))^⋆ * H * a * top ≤ (H * (d ⊓ - a))^⋆ * H * a * top
      using star.left-plus-below-circ mult-left-isotone by simp
    have 52: (H * (d ⊓ - a) ⊔ H * (d ⊓ - a)^T) * (H * (d ⊓ - a))^⋆ * H * a * top = H * (d ⊓ - a)
* (H * (d ⊓ - a))^⋆ * H * a * top ⊔ H * (d ⊓ - a)^T * (H * (d ⊓ - a))^⋆  * H * a * top
      using mult-right-dist-sup by auto
    then have ... ≤ (H * (d ⊓ - a))^⋆ * H * a * top ⊔ H * (d ⊓ - a)^T * (H * (d ⊓ - a))^⋆  * H * a
* top
      using star.left-plus-below-circ mult-left-isotone sup-left-isotone by auto
    thus ?thesis
      using 4 51 52 mult-assoc by auto
  qed
  then have (H * (d ⊓ - a) ⊔ H * (d ⊓ - a)^T)^⋆ * H * a * top ≤ ?lhs
  proof -
    have (H * (d ⊓ - a) ⊔ H * (d ⊓ - a)^T)^⋆ * (H * (d ⊓ - a))^⋆  * H * a * top ≤ ?lhs
      using 5 star-left-induct-mult-iff mult-assoc by auto
    thus ?thesis
      using star.circ-decompose-11 star-decompose-1 by auto
  qed
  then have 6: (H * ((d ⊓ - a) ⊔ (d ⊓ - a)^T))^⋆ * H * a * top ≤ ?lhs
    using mult-left-dist-sup by auto
  have 7: (H * (d ⊓ - a))^⋆ * H * a * top ≤ (H * ((d ⊓ - a) ⊔ (d ⊓ - a)^T))^⋆ * H * a * top
    by (simp add: mult-left-isotone semiring.distrib-left star-isotone)
  thus ?thesis
    using 6 7 by (simp add: mult-assoc)
qed

end
```

B.2.2 An operation to select components

```
class stone-kleene-relation-algebra-tarski = stone-kleene-relation-algebra +
  assumes tarski: regular x ⟹ x ≠ bot ⟹ top * x * top = top
begin
end
```

We introduce the operation *choose-component*. Axiom *component-in-v* expresses that the result of *choose-component* is contained in the set of vertices, v, we are selecting from, ignoring

the weights. Axiom *component-is-vector* states that the result of *choose-component* is a vector. Axiom *component-is-regular* states that the result of *choose-component* is regular. Axiom *component-is-connected* states that any two vertices from the result of *choose-component* are connected in e. Axiom *component-single* states that the result of *choose-component* is closed under being connected in e. Axiom *component-not-bot-when-v-bot-bot* states that *choose-component* returns a non-empty component if the input satisfies the given criteria.

class *choose-component-algebra* = *stone-relation-algebra* +
  fixes *choose-component* :: $'a \Rightarrow 'a \Rightarrow 'a$
  assumes *component-in-v*: *choose-component e v* $\leq --v$
  assumes *component-is-vector*: *vector* (*choose-component e v*)
  assumes *component-is-regular*: *regular* (*choose-component e v*)
  assumes *component-is-connected*: *choose-component e v* $*$ (*choose-component e v*)$^T \leq e$
  assumes *component-single*: *choose-component e v* $=$ *e* $*$ *choose-component e v*
  assumes *component-not-bot-when-v-bot-bot*:
    regular e
  $\wedge$ *equivalence e*
  $\wedge$ *vector v*
  $\wedge$ *regular v*
  $\wedge$ *e* $*$ *v* $=$ *v*
  $\wedge$ *v* $\neq$ *bot* $\longrightarrow$ *choose-component e v* $\neq$ *bot*

Theorem 1

We show that *m-kleene-algebras* form an instance of *choose-component-algebra* when the *choose-component* operation is defined as follows:

context *m-kleene-algebra*
begin
definition *choose-component-input-condition e v* $\equiv$
    regular e
  $\wedge$ *equivalence e*
  $\wedge$ *vector v*
  $\wedge$ *regular v*
  $\wedge$ *e* $*$ *v* $=$ *v*

definition *m-choose-component e v* $\equiv$
  if choose-component-input-condition e v then
    e $*$ *minarc*(*v*) $*$ *top*
  else
    bot

sublocale *m-choose-component-algebra*: *choose-component-algebra* **where** *choose-component* $=$ *m-choose-component*
proof (*unfold-locales*)
  fix *e v*
  show *m-choose-component e v* $\leq --$ *v*
  proof (*cases choose-component-input-condition e v*)
    case *True*
    then have *m-choose-component e v* $=$ *e* $*$ *minarc*(*v*) $*$ *top*
      by (*simp add*: *m-choose-component-def*)
    also have ... $\leq$ *e* $*$ $--$*v* $*$ *top*
      by (*simp add*: *comp-isotone minarc-below*)
    also have ... $=$ *e* $*$ *v* $*$ *top*
      using *True choose-component-input-condition-def* **by** *auto*
    also have ... $=$ *v* $*$ *top*
      using *True choose-component-input-condition-def* **by** *auto*
    finally show *?thesis*
      using *True choose-component-input-condition-def* **by** *auto*
  next

```
    case False
    then have m-choose-component e v = bot
      using False m-choose-component-def by auto
    thus ?thesis
      by simp
  qed
next
  fix e v
  show vector (m-choose-component e v)
  proof (cases choose-component-input-condition e v)
    case True
    thus ?thesis
      by (simp add: mult-assoc m-choose-component-def)
  next
    case False
    thus ?thesis
      by (simp add: m-choose-component-def)
  qed
next
  fix e v
  show regular (m-choose-component e v)
    using choose-component-input-condition-def minarc-regular regular-closed-star regular-mult-closed
m-choose-component-def by auto
next
  fix e v
  show m-choose-component e v * (m-choose-component e v)^T ≤ e
  proof (cases choose-component-input-condition e v)
    case True
    assume 1: choose-component-input-condition e v
    then have m-choose-component e v * (m-choose-component e v)^T  = e * minarc(v) * top * (e *
minarc(v) * top)^T
      by (simp add: m-choose-component-def)
    also have ... = e * minarc(v) * top * top^T * minarc(v)^T * e^T
      using comp-associative conv-dist-comp by presburger
    also have ... = e * minarc(v) * top * top * minarc(v)^T * e
      using 1 choose-component-input-condition-def by auto
    also have ... = e * minarc(v) * top * minarc(v)^T * e
      by (simp add: comp-associative)
    also have ... ≤ e
    proof (cases v = bot)
      case True
      thus ?thesis
        by (simp add: True minarc-bot)
    next
      case False
      assume 3: v ≠ bot
      then have e * minarc(v) * top * minarc(v)^T ≤ e * 1
        using 3 minarc-arc arc-expanded comp-associative mult-right-isotone by fastforce
      then have e * minarc(v) * top * minarc(v)^T * e ≤ e * 1 * e
        using mult-left-isotone by auto
      also have ... = e
        using 1 choose-component-input-condition-def preorder-idempotent by auto
      thus ?thesis
        using calculation by auto
    qed
    thus ?thesis
      by (simp add: calculation)
  next
```

```
    case False
    thus ?thesis
      by (simp add: m-choose-component-def)
  qed
next
  fix e v
  show m-choose-component e v = e ∗ m-choose-component e v
  proof (cases choose-component-input-condition e v)
    case True
    thus ?thesis
      by (metis choose-component-input-condition-def preorder-idempotent m-choose-component-def
mult-assoc)
  next
    case False
    thus ?thesis
      by (simp add: m-choose-component-def)
  qed
next
  fix e v
  show regular e ∧ equivalence e ∧ vector v ∧ regular v ∧ e ∗ v = v ∧ v ≠ bot ⟶
m-choose-component e v ≠ bot
  proof (cases choose-component-input-condition e v)
    case True
    then have m-choose-component e v ≥ minarc(v) ∗ top
      by (metis choose-component-input-condition-def mult-1-left mult-left-isotone
m-choose-component-def)
    also have ... ≥ minarc(v)
      using calculation dual-order.trans top-right-mult-increasing by blast
    thus ?thesis
      using True bot-unique minarc-bot-iff by fastforce
  next
    case False
    thus ?thesis
      using choose-component-input-condition-def by blast
  qed
qed

end
```

B.2.3 m-k-Stone-Kleene relation algebras

```
class m-kleene-algebra-tarski =
  m-kleene-algebra
  + stone-relation-algebra-tarski
  + choose-component-algebra
begin

abbreviation selected-edge h j g ≡ minarc (choose-component (forest-components h) j ∗ −
choose-component (forest-components h) j^T ⊓ g)
abbreviation path f h j g ≡ top ∗ selected-edge h j g ∗ (f ⊓ − selected-edge h j g^T)^T⋆

definition boruvka-outer-invariant f g ≡
    symmetric g
  ∧ forest f
  ∧ f ≤ −−g
  ∧ regular f
  ∧ (∃ w . minimum-spanning-forest w g ∧ f ≤ w ⊔ w^T)
```

definition *boruvka-inner-invariant j f h g d* ≡
  boruvka-outer-invariant f g
 ∧ *g* ≠ *bot*
 ∧ *vector j*
 ∧ *regular j*
 ∧ *boruvka-outer-invariant h g*
 ∧ *forest h*
 ∧ *forest-components h* ≤ *forest-components f*
 ∧ *big-forest* (*forest-components h*) *d*
 ∧ *d* ∗ *top* ≤ − *j*
 ∧ *forest-components h* ∗ *j* = *j*
 ∧ *forest-components f* = (*forest-components h* ∗ (*d* ⊔ d^T))* ∗ *forest-components h*
 ∧ *f* ⊔ f^T = *h* ⊔ h^T ⊔ *d* ⊔ d^T
 ∧ (∀ *a b* . *bf-between-arcs a b* (*forest-components h*) *d* ∧ *a* ≤ −(*forest-components h*) ⊓ −− *g* ∧ *b* ≤ *d*
  ⟶ *sum*(*b* ⊓ *g*) ≤ *sum*(*a* ⊓ *g*))
 ∧ *regular d*

lemma *expression-equivalent-without-e-complement*:
 assumes *selected-edge h j g* ≤ − *forest-components f*
 shows *f* ⊓ − (*selected-edge h j g*)^T ⊓ − (*path f h j g*) ⊔ (*f* ⊓ − (*selected-edge h j g*)^T ⊓ (*path f h j g*))^T ⊔ (*selected-edge h j g*)
    = *f* ⊓ − (*path f h j g*) ⊔ (*f* ⊓ (*path f h j g*))^T ⊔ (*selected-edge h j g*)
proof −
 let *?p* = *path f h j g*
 let *?e* = *selected-edge h j g*
 let *?F* = *forest-components f*
 have *1*: *?e* ≤ − *?F*
  by (*simp add*: *assms*)
 have f^T ≤ *?F*
  by (*metis conv-dist-comp conv-involutive conv-order conv-star-commute forest-components-increasing*)
 then have − *?F* ≤ − f^T
  using *p-antitone* **by** *auto*
 then have *?e* ≤ − f^T
  using *1 dual-order.trans* **by** *blast*
 then have f^T ≤ − *?e*
  by (*simp add*: *p-antitone-iff*)
 then have f^{TT} ≤ − $?e^T$
  by (*metis conv-complement conv-dist-inf inf.orderE inf.orderI*)
 then have *f* ≤ − $?e^T$
  by *auto*
 then have *f* = *f* ⊓ − $?e^T$
  using *inf.orderE* **by** *blast*
 then have *f* ⊓ − $?e^T$ ⊓ − *?p* ⊔ (*f* ⊓ − $?e^T$ ⊓ *?p*)^T ⊔ *?e* = *f* ⊓ − *?p* ⊔ (*f* ⊓ *?p*)^T ⊔ *?e*
  by *auto*
 thus *?thesis* **by** *auto*
qed

Theorem 2

lemma *et-below-j*:
 assumes *vector j*
  and *regular j*
  and *j* ≠ *bot*
 shows *selected-edge h j g* ∗ *top* ≤ *j*
proof −
 let *?e* = *selected-edge h j g*
 let *?c* = *choose-component* (*forest-components h*) *j*

```
  have ?e * top ≤ --(?c * -?c^T ⊓ g) * top
    using comp-isotone minarc-below by blast
  also have ... = (--(?c * -?c^T) ⊓ --g) * top
    by simp
  also have ... = (?c * -?c^T ⊓ --g) * top
    using component-is-regular regular-mult-closed by auto
  also have ... = (?c ⊓ -?c^T ⊓ --g) * top
    using assms(1, 2, 3) component-is-vector conv-complement vector-complement-closed
vector-covector by metis
  also have ... ≤ ?c * top
    using inf.cobounded1 mult-left-isotone order-trans by blast
  also have ... ≤ j * top
    by (metis assms(2) comp-inf.star.circ-sup-2 comp-isotone component-in-v)
  also have ... = j
    by (simp add: assms(1))
  finally show ?thesis
    by simp
qed

lemma fc-j-eq-j-inv:
  assumes forest h
    and forest-components h * j = j
  shows forest-components h * (j ⊓ - choose-component (forest-components h) j) = j ⊓ -
choose-component (forest-components h) j
proof -
  let ?c = choose-component (forest-components h) j
  let ?H = forest-components h
  have 1:equivalence ?H
    by (simp add: assms(1) forest-components-equivalence)
  have ?H * (j ⊓ - ?c) = ?H * j ⊓ ?H * - ?c
    by (metis 1 assms(2) equivalence-comp-dist-inf inf.sup-monoid.add-commute)
  then have 2: ?H * (j ⊓ - ?c) = j ⊓ ?H * - ?c
    by (simp add: assms(2))
  have 3: j ⊓ - ?c ≤ ?H * - ?c
    by (metis 1 assms(2) dedekind-1 dual-order.trans equivalence-comp-dist-inf inf.cobounded2)
  have ?H * ?c ≤ ?c
    using component-single by auto
  then have ?H^T * ?c ≤ ?c
    using 1 by simp
  then have ?H * - ?c ≤ - ?c
    using component-is-regular schroeder-3-p by force
  then have j ⊓ ?H * - ?c ≤ j ⊓ - ?c
    using inf.sup-right-isotone by auto
  thus ?thesis
    using 2 3 antisym by simp
qed
```

Theorem 5

```
lemma big-forest-path-split-disj:
  assumes equivalence H
    and arc c
    and regular a ∧ regular b ∧ regular c ∧ regular d ∧ regular H
  shows bf-between-arcs a b H (d ⊔ c) ⟷ bf-between-arcs a b H d ∨ (bf-between-arcs a c H d ∧
bf-between-arcs c b H d)
proof -
  have 1: bf-between-arcs a b H (d ⊔ c) ⟶ bf-between-arcs a b H d ∨ (bf-between-arcs a c H d ∧
bf-between-arcs c b H d)
  proof (rule impI)
```

assume *11*: *bf-between-arcs a b H* $(d \sqcup c)$
then have $a^T * top \leq (H * (d \sqcup c))^\star * H * b * top$
by (*simp add*: *bf-between-arcs-def*)
also have $... = ((H * d)^\star \sqcup (H * d)^\star * H * c * (H * d)^\star) * H * b * top$
using *assms*(*1*, *2*) *path-through-components* **by** *simp*
also have $... = (H * d)^\star * H * b * top \sqcup (H * d)^\star * H * c * (H * d)^\star * H * b * top$
by (*simp add*: *mult-right-dist-sup*)
finally have *12*:$a^T * top \leq (H * d)^\star * H * b * top \sqcup (H * d)^\star * H * c * (H * d)^\star * H * b * top$
by *simp*
have *13*: $a^T * top \leq (H * d)^\star * H * b * top \vee a^T * top \leq (H * d)^\star * H * c * (H * d)^\star * H * b * top$
proof (*rule point-in-vector-sup*)
show *point* $(a^T * top)$
using *11 bf-between-arcs-def mult-assoc* **by** *auto*
next
show *vector* $((H * d)^\star * H * b * top)$
using *vector-mult-closed* **by** *simp*
next
show *regular* $((H * d)^\star * H * b * top)$
using *assms*(*3*) *pp-dist-comp pp-dist-star* **by** *auto*
next
show $a^T * top \leq (H * d)^\star * H * b * top \sqcup (H * d)^\star * H * c * (H * d)^\star * H * b * top$
using *12* **by** *simp*
qed
thus *bf-between-arcs a b H d* $\vee$ (*bf-between-arcs a c H d* $\wedge$ *bf-between-arcs c b H d*)
proof (*cases* $a^T * top \leq (H * d)^\star * H * b * top$)
case *True*
assume $a^T * top \leq (H * d)^\star * H * b * top$
then have *bf-between-arcs a b H d*
using *11 bf-between-arcs-def* **by** *auto*
thus *?thesis*
by *simp*
next
case *False*
have *14*: $a^T * top \leq (H * d)^\star * H * c * (H * d)^\star * H * b * top$
using *13* **by** (*simp add*: *False*)
then have *15*: $a^T * top \leq (H * d)^\star * H * c * top$
by (*metis mult-right-isotone order-lesseq-imp top-greatest mult-assoc*)
have $c^T * top \leq (H * d)^\star * H * b * top$
proof (*rule ccontr*)
assume $\neg\ c^T * top \leq (H * d)^\star * H * b * top$
then have $c^T * top \leq -((H * d)^\star * H * b * top)$
by (*meson assms*(*2*, *3*) *point-in-vector-or-complement regular-closed-star regular-closed-top regular-mult-closed vector-mult-closed vector-top-closed*)
then have $c * (H * d)^\star * H * b * top \leq bot$
using *schroeder-3-p mult-assoc* **by** *auto*
thus *False*
using *13 False sup.absorb-iff1 mult-assoc* **by** *auto*
qed
then have *bf-between-arcs a c H d* $\wedge$ *bf-between-arcs c b H d*
using *11 15 assms*(*2*) *bf-between-arcs-def* **by** *auto*
thus *?thesis*
by *simp*
qed
qed
have *2*: *bf-between-arcs a b H d* $\vee$ (*bf-between-arcs a c H d* $\wedge$ *bf-between-arcs c b H d*) $\longrightarrow$ *bf-between-arcs a b H* $(d \sqcup c)$
proof −

```
    have 21: bf-between-arcs a b H d ⟶ bf-between-arcs a b H (d ⊔ c)
    proof (rule impI)
      assume 22:bf-between-arcs a b H d
      then have a^T * top ≤ (H * d)^⋆ * H * b * top
        using bf-between-arcs-def by blast
      then have a^T * top ≤ (H * (d ⊔ c))^⋆ * H * b * top
        by (simp add: mult-left-isotone mult-right-dist-sup mult-right-isotone order.trans star-isotone
star-slide)
      thus bf-between-arcs a b H (d ⊔ c)
        using 22 bf-between-arcs-def by blast
    qed
    have bf-between-arcs a c H d ∧ bf-between-arcs c b H d ⟶ bf-between-arcs a b H (d ⊔ c)
    proof (rule impI)
      assume 23: bf-between-arcs a c H d ∧ bf-between-arcs c b H d
      then have a^T * top ≤ (H * d)^⋆ * H * c * top
        using bf-between-arcs-def by blast
      also have ... ≤ (H * d)^⋆ * H * c * c^T * c * top
        using ex231c by (metis comp-inf.star.circ-sup-2 mult-isotone mult-right-isotone mult-assoc)
      also have ... ≤ (H * d)^⋆ * H * c * c^T * top
        by (simp add: mult-right-isotone mult-assoc)
      also have ... ≤ (H * d)^⋆ * H * c * (H * d)^⋆ * H * b * top
        using 23 mult-right-isotone mult-assoc by (simp add: bf-between-arcs-def)
      also have ... ≤ (H * d)^⋆ * H * b * top ⊔ (H * d)^⋆ * H * c * (H * d)^⋆ * H * b * top
        by (simp add: bf-between-arcs-def)
      finally have a^T * top ≤ (H * (d ⊔ c))^⋆ * H * b * top
        using assms(1, 2) path-through-components mult-right-dist-sup by simp
      thus bf-between-arcs a b H (d ⊔ c)
        using 23 bf-between-arcs-def by blast
    qed
    thus ?thesis
      using 21 by auto
  qed
  thus ?thesis
    using 1 2 by blast
qed

lemma dT-He-eq-bot:
  assumes vector j
    and regular j
    and d * top ≤ - j
    and forest-components h * j = j
    and j ≠ bot
  shows d^T * forest-components h * selected-edge h j g ≤ bot
proof -
  let ?e = selected-edge h j g
  let ?H = forest-components h
  have 1: ?e * top ≤ j
    using assms(1, 2, 5) et-below-j by auto
  have d^T * ?H * ?e ≤ (d * top)^T * ?H * (?e * top)
    by (simp add: comp-isotone conv-isotone top-right-mult-increasing)
  also have ... ≤ (d * top)^T * ?H * j
    using 1 mult-right-isotone by auto
  also have ... ≤ (- j)^T * ?H * j
    by (simp add: assms(3) conv-isotone mult-left-isotone)
  also have ... = (- j)^T * j
    using assms(4) comp-associative by auto
  also have ... = bot
    by (simp add: assms(1) conv-complement covector-vector-comp)
```

```
  finally show ?thesis
    using coreflexive-bot-closed le-bot by blast
qed

lemma big-forest-d-U-e:
  assumes forest f
    and vector j
    and regular j
    and forest h
    and forest-components h ≤ forest-components f
    and big-forest (forest-components h) d
    and d * top ≤ - j
    and forest-components h * j = j
    and f ⊔ f^T = h ⊔ h^T ⊔ d ⊔ d^T
    and selected-edge h j g ≤ - forest-components f
    and selected-edge h j g ≠ bot
    and j ≠ bot
  shows big-forest (forest-components h) (d ⊔ selected-edge h j g)
proof (unfold big-forest-def, intro conjI)
  let ?H = forest-components h
  let ?F = forest-components f
  let ?e = selected-edge h j g
  let ?d' = d ⊔ ?e
  show 01: reflexive ?H
    by (simp add: assms(4) forest-components-equivalence)
  show 02: transitive ?H
    by (simp add: assms(4) forest-components-equivalence)
  show 03: symmetric ?H
    by (simp add: assms(4) forest-components-equivalence)
  have 04: equivalence ?H
    by (simp add: 01 02 03)
  show 1: ?d' ≤ - ?H
  proof -
    have ?H ≤ ?F
      by (simp add: assms(5))
    then have 11: ?e ≤ - ?H
      using assms(10) order-lesseq-imp p-antitone by blast
    have d ≤ - ?H
      using assms(6) big-forest-def by auto
    thus ?thesis
      by (simp add: 11)
  qed
  show univalent (?H * ?d')
  proof -
    have (?H * ?d')^T * (?H * ?d') = ?d'^T * ?H^T * ?H * ?d'
      using conv-dist-comp mult-assoc by auto
    also have ... = ?d'^T * ?H * ?H * ?d'
      by (simp add: conv-dist-comp conv-star-commute)
    also have ... = ?d'^T * ?H * ?d'
      by (metis 01 02 preorder-idempotent mult-assoc)
    finally have 21: univalent (?H * ?d') ⟷ ?e^T * ?H * d ⊔ d^T * ?H * ?e ⊔ ?e^T * ?H * ?e ⊔ d^T
* ?H * d ≤ 1
      using conv-dist-sup semiring.distrib-left semiring.distrib-right by auto
    have 22: ?e^T * ?H * ?e ≤ 1
    proof -
      have 221: ?e^T * ?H * ?e ≤ ?e^T * top * ?e
        by (simp add: mult-left-isotone mult-right-isotone)
      have arc ?e
```

```
      using assms(11) minarc-arc minarc-bot-iff by blast
    then have ?e^T * top * ?e ≤ 1
      using arc-expanded by blast
    thus ?thesis
      using 221 dual-order.trans by blast
  qed
  have 24: d^T * ?H * ?e ≤ 1
    by (metis assms(2, 3, 7, 8, 12) dT-He-eq-bot coreflexive-bot-closed le-bot)
  then have (d^T * ?H * ?e)^T ≤ 1^T
    using conv-isotone by blast
  then have ?e^T * ?H^T * d^TT ≤ 1
    by (simp add: conv-dist-comp mult-assoc)
  then have 25: ?e^T * ?H * d ≤ 1
    using assms(4) fch-equivalence by auto
  have 8: d^T * ?H * d ≤ 1
    using 04 assms(6) dTransHd-le-1 big-forest-def by blast
  thus ?thesis
    using 21 22 24 25 by simp
qed
show coreflexive (?H ⊓ ?d' * ?d'^T)
proof -
  have coreflexive (?H ⊓ ?d' * ?d'^T) ⟷ ?H ⊓ (d ⊔ ?e) * (d^T ⊔ ?e^T) ≤ 1
    by (simp add: conv-dist-sup)
  also have ... ⟷ ?H ⊓ (d * d^T ⊔ d * ?e^T ⊔ ?e * d^T ⊔ ?e * ?e^T) ≤ 1
    by (metis mult-left-dist-sup mult-right-dist-sup sup.left-commute sup-commute)
  finally have 1: coreflexive (?H ⊓ ?d' * ?d'^T) ⟷ ?H ⊓ d * d^T ⊔ ?H ⊓ d * ?e^T ⊔ ?H ⊓ ?e * d^T
⊔ ?H ⊓ ?e * ?e^T ≤ 1
    by (simp add: inf-sup-distrib1)
  have 31: ?H ⊓ d * d^T ≤ 1
    using assms(6) big-forest-def by blast
  have 32: ?H ⊓ ?e * d^T ≤ 1
  proof -
    have ?e * d^T ≤ ?e * top * (d * top)^T
      by (simp add: conv-isotone mult-isotone top-right-mult-increasing)
    also have ... ≤ ?e * top * - j^T
      by (metis assms(7) conv-complement conv-isotone mult-right-isotone)
    also have ... ≤ j * - j^T
      using assms(2, 3, 12) et-below-j mult-left-isotone by auto
    also have ... ≤ - ?H
      by (metis 03 assms(2, 3, 8) conv-complement conv-dist-comp equivalence-top-closed
mult-left-isotone schroeder-3-p vector-top-closed)
    finally have ?e * d^T ≤ - ?H
      by simp
    thus ?thesis
      by (metis inf.coboundedI1 p-antitone-iff p-shunting-swap regular-one-closed)
  qed
  have 33: ?H ⊓ d * ?e^T ≤ 1
  proof -
    have 331: injective h
      by (simp add: assms(4))
    have (?H ⊓ ?e * d^T)^T ≤ 1
      using 32 coreflexive-conv-closed by auto
    then have ?H ⊓ (?e * d^T)^T ≤ 1
      using 331 conv-dist-inf forest-components-equivalence by auto
    thus ?thesis
      using conv-dist-comp by auto
  qed
  have 34: ?H ⊓ ?e * ?e^T ≤ 1
```

```
    proof −
      have 341:arc ?e ∧ arc (?e^T)
        using assms(11) minarc-arc minarc-bot-iff by auto
      have ?H ⊓ ?e ∗ ?e^T ≤ ?e ∗ ?e^T
        by auto
      thus ?thesis
        using 341 arc-injective le-infI2 by blast
    qed
    thus ?thesis
      using 1 31 32 33 34 by simp
  qed
  show 4:(?H ∗ (d ⊔ ?e))^+ ≤ − ?H
  proof −
    have ?e ≤ − ?F
      by (simp add: assms(10))
    then have ?F ≤ − ?e
      by (simp add: p-antitone-iff)
    then have ?F^T ∗ ?F ≤ − ?e
      using assms(1) fch-equivalence by fastforce
    then have ?F^T ∗ ?F ∗ ?F^T ≤ − ?e
      by (metis assms(1) fch-equivalence forest-components-star star.circ-decompose-9)
    then have 41: ?F ∗ ?e ∗ ?F ≤ − ?F
      using triple-schroeder-p by blast
    then have 42:(?F ∗ ?F)^⋆ ∗ ?F ∗ ?e ∗ (?F ∗ ?F)^⋆ ≤ − ?F
    proof −
      have 43: ?F ∗ ?F = ?F
        using assms(1) forest-components-equivalence preorder-idempotent by auto
      then have ?F ∗ ?e ∗ ?F = ?F ∗ ?F ∗ ?e ∗ ?F
        by simp
      also have ... = (?F)^⋆ ∗ ?F ∗ ?e ∗ (?F)^⋆
        by (simp add: assms(1) forest-components-star)
      also have ... = (?F ∗ ?F)^⋆ ∗ ?F ∗ ?e ∗ (?F ∗ ?F)^⋆
        using 43 by simp
      finally show ?thesis
        using 41 by simp
    qed
    then have 44: (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ≤ − ?H
    proof −
      have 45: ?H ≤ ?F
        by (simp add: assms(5))
      then have 46:?H ∗ ?e ≤ ?F ∗ ?e
        by (simp add: mult-left-isotone)
      have d ≤ f ⊔ f^T
        using assms(9) sup.left-commute sup-commute by auto
      also have ... ≤ ?F
        by (metis forest-components-increasing le-supI2 star.circ-back-loop-fixpoint star.circ-increasing
sup.bounded-iff)
      finally have d ≤ ?F
        by simp
      then have ?H ∗ d ≤ ?F ∗ ?F
        using 45 mult-isotone by auto
      then have 47: (?H ∗ d)^⋆ ≤ (?F ∗ ?F)^⋆
        by (simp add: star-isotone)
      then have (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ≤ (?H ∗ d)^⋆ ∗ ?F ∗ ?e ∗ (?H ∗ d)^⋆
        using 46 by (metis mult-left-isotone mult-right-isotone mult-assoc)
      also have ... ≤ (?F ∗ ?F)^⋆ ∗ ?F ∗ ?e ∗ (?F ∗ ?F)^⋆
        using 47 mult-left-isotone mult-right-isotone by (simp add: comp-isotone)
      also have ... ≤ − ?F
```

```
      using 42 by simp
    also have ... ≤ − ?H
      using 45 by (simp add: p-antitone)
    finally show ?thesis
      by simp
  qed
  have (?H ∗ (d ⊔ ?e))⁺ = (?H ∗ (d ⊔ ?e))^⋆ ∗ (?H ∗ (d ⊔ ?e))
    using star.circ-plus-same by auto
  also have ... = ((?H ∗ d)^⋆ ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆) ∗ (?H ∗ (d ⊔ ?e))
    using assms(4, 11) forest-components-equivalence minarc-arc minarc-bot-iff
path-through-components by auto
  also have ... = (?H ∗ d)^⋆ ∗ (?H ∗ (d ⊔ ?e)) ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗ (?H ∗ (d ⊔ ?e))
    using mult-right-dist-sup by auto
  also have ... = (?H ∗ d)^⋆ ∗ (?H ∗ d ⊔ ?H ∗ ?e) ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗ (?H ∗ d ⊔
?H ∗ ?e)
    by (simp add: mult-left-dist-sup)
  also have ... = (?H ∗ d)^⋆ ∗ ?H ∗ d ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗
(?H ∗ d ⊔ ?H ∗ ?e)
    using mult-left-dist-sup mult-assoc by auto
  also have ... = (?H ∗ d)⁺ ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗ (?H ∗ d
⊔ ?H ∗ ?e)
    by (simp add: star.circ-plus-same mult-assoc)
  also have ... = (?H ∗ d)⁺ ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗ ?H ∗ d ⊔
(?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗ ?H ∗ ?e
    by (simp add: mult.semigroup-axioms semiring.distrib-left sup.semigroup-axioms semigroup.assoc)
  also have ... ≤ (?H ∗ d)⁺ ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗ ?H ∗ d ⊔
(?H ∗ d)^⋆ ∗ ?H ∗ ?e
  proof −
    have ?e ∗ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ≤ ?e ∗ top ∗ ?e
      by (metis comp-associative comp-inf.coreflexive-idempotent comp-inf.coreflexive-transitive
comp-isotone top.extremum)
    also have ... ≤ ?e
      using assms(11) arc-top-arc minarc-arc minarc-bot-iff by auto
    finally have ?e ∗ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ≤ ?e
      by simp
    then have (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ≤ (?H ∗ d)^⋆ ∗ ?H ∗ ?e
      by (simp add: comp-associative comp-isotone)
    thus ?thesis
      using sup-right-isotone by blast
  qed
  also have ... = (?H ∗ d)⁺ ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆ ∗ ?H ∗ d
    by (smt eq-iff sup.left-commute sup.orderE sup-commute)
  also have ... = (?H ∗ d)⁺ ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)⁺
    using star.circ-plus-same mult-assoc by auto
  also have ... = (?H ∗ d)⁺ ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (1 ⊔ (?H ∗ d)⁺)
    by (simp add: mult-left-dist-sup sup-assoc)
  also have ... = (?H ∗ d)⁺ ⊔ (?H ∗ d)^⋆ ∗ ?H ∗ ?e ∗ (?H ∗ d)^⋆
    by (simp add: star-left-unfold-equal)
  also have ... ≤ − ?H
    using 44 assms(6) big-forest-def by auto
  finally show ?thesis
    by simp
 qed
qed

lemma shows-arc-x:
 assumes big-forest H d
   and bf-between-arcs a e H d
```

```
    and H * d * (H * d)^* ≤ - H
    and ¬ a^T * top ≤ H * e * top
    and regular a
    and regular e
    and regular H
    and regular d
  shows arc (d ⊓ top * e^T * H ⊓ (H * d^T)^* * H * a^T * top)
proof -
  let ?x = d ⊓ top * e^T * H ⊓ (H * d^T)^* * H * a^T * top
  have 1:regular ?x
    using assms(5, 6, 7, 8) regular-closed-star regular-conv-closed regular-mult-closed by auto
  have 2: a^T * top * a ≤ 1
    using arc-expanded assms(2) bf-between-arcs-def by auto
  have 3: e * top * e^T ≤ 1
    using assms(2) bf-between-arcs-def arc-expanded by blast
  have 4: top * ?x * top = top
  proof -
    have a^T * top ≤ (H * d)^* * H * e * top
      using assms(2) bf-between-arcs-def by blast
    also have ... = H * e * top ⊔ (H * d)^* * H * d * H * e * top
      by (metis star.circ-loop-fixpoint star.circ-plus-same sup-commute mult-assoc)
    finally have a^T * top ≤ H * e * top ⊔ (H * d)^* * H * d * H * e * top
      by simp
    then have a^T * top ≤ H * e * top ∨ a^T * top ≤ (H * d)^* * H * d * H * e * top
      using assms(2, 6, 7) point-in-vector-sup bf-between-arcs-def regular-mult-closed
vector-mult-closed by auto
    then have a^T * top ≤ (H * d)^* * H * d * H * e * top
      using assms(4) by blast
    also have ... = (H * d)^* * H * d * (H * e * top ⊓ H * e * top)
      by (simp add: mult-assoc)
    also have ... = (H * d)^* * H * (d ⊓ (H * e * top)^T) * H * e * top
      by (metis comp-associative covector-inf-comp-3 star.circ-left-top star.circ-top)
    also have ... = (H * d)^* * H * (d ⊓ top^T * e^T * H^T) * H * e * top
      using conv-dist-comp mult-assoc by auto
    also have ... = (H * d)^* * H * (d ⊓ top * e^T * H) * H * e * top
      using assms(1) by (simp add: big-forest-def)
    finally have 2: a^T * top ≤ (H * d)^* * H * (d ⊓ top * e^T * H) * H * e * top
      by simp
    then have e * top ≤ ((H * d)^* * H * (d ⊓ top * e^T * H) * H)^T * a^T * top
    proof -
      have bijective (e * top) ∧ bijective (a^T * top)
        using assms(2) bf-between-arcs-def by auto
      thus ?thesis
        using 2 bijective-reverse mult-assoc by metis
    qed
    also have ... = H^T * (d ⊓ top * e^T * H)^T * H^T * (H * d)^{*T} * a^T * top
      by (simp add: conv-dist-comp mult-assoc)
    also have ... = H * (d ⊓ top * e^T * H)^T * H * (H * d)^{*T} * a^T * top
      using assms(1) big-forest-def by auto
    also have ... = H * (d ⊓ top * e^T * H)^T * H * (d^T * H)^* * a^T * top
      using assms(1) big-forest-def conv-dist-comp conv-star-commute by auto
    also have ... = H * (d^T ⊓ H * e * top) * H * (d^T * H)^* * a^T * top
      using assms(1) conv-dist-comp big-forest-def comp-associative conv-dist-inf by auto
    also have ... = H * (d^T ⊓ H * e * top) * (H * d^T)^* * H * a^T * top
      by (simp add: comp-associative star-slide)
    also have ... = H * (d^T ⊓ H * e * top) * ((H * d^T)^* * H * a^T * top ⊓ (H * d^T)^* * H * a^T * top)
      using mult-assoc by auto
    also have ... = H * (d^T ⊓ H * e * top ⊓ ((H * d^T)^* * H * a^T * top)^T) * (H * d^T)^* * H * a^T *
```

top
by (*smt comp-inf-vector covector-comp-inf vector-conv-covector vector-top-closed mult-assoc*)
also have ... = $H * (d^T \sqcap (top * e^T * H)^T \sqcap ((H * d^T)^\star * H * a^T * top)^T) * (H * d^T)^\star * H * a^T * top$
using *assms(1) big-forest-def conv-dist-comp mult-assoc* **by** *auto*
also have ... = $H * (d \sqcap top * e^T * H \sqcap (H * d^T)^\star * H * a^T * top)^T * (H * d^T)^\star * H * a^T * top$
by (*simp add*: *conv-dist-inf*)
finally have *3*: $e * top \leq H * ?x^T * (H * d^T)^\star * H * a^T * top$
by *auto*
have $?x \neq bot$
proof (*rule ccontr*)
assume $\neg\ ?x \neq bot$
then have $e * top = bot$
using *3 le-bot* **by** *auto*
thus *False*
using *assms(2, 4) bf-between-arcs-def mult-assoc semiring.mult-zero-right* **by** *auto*
qed
thus *?thesis*
using *1* **using** *tarski* **by** *blast*
qed
have *5*: $?x * top * ?x^T \leq 1$
proof −
have *51*: $H * (d * H)^\star \sqcap d * H * d^T \leq 1$
proof −
have *511*: $d * (H * d)^\star \leq - H$
using *assms(1, 3) big-forest-def preorder-idempotent schroeder-4-p triple-schroeder-p* **by** *fastforce*
then have $(d * H)^\star * d \leq - H$
using *star-slide* **by** *auto*
then have $H * (d^T * H)^\star \leq - d$
using *assms(1) big-forest-def conv-dist-comp conv-star-commute schroeder-4-p* **by** (*smt star-slide*)
then have $H * (d * H)^\star \leq - d^T$
by (*metis 511 assms(1) big-forest-def schroeder-5-p star-slide*)
then have $H * (d * H)^\star \leq - (H * d^T)$
by (*metis assms(3) p-antitone-iff schroeder-4-p star-slide mult-assoc*)
then have $H * (d * H)^\star \sqcap H * d^T \leq bot$
by (*simp add*: *bot-unique pseudo-complement*)
then have $H * d * (H * (d * H)^\star \sqcap H * d^T) \leq 1$
by (*simp add*: *bot-unique*)
then have *512*: $H * d * H * (d * H)^\star \sqcap H * d * H * d^T \leq 1$
using *univalent-comp-left-dist-inf assms(1) big-forest-def mult-assoc* **by** *fastforce*
then have *513*: $H * d * H * (d * H)^\star \sqcap d * H * d^T \leq 1$
proof −
have $d * H * d^T \leq H * d * H * d^T$
by (*metis assms(1) big-forest-def conv-dist-comp conv-involutive mult-1-right mult-left-isotone*)
thus *?thesis*
by (*smt 512 dual-order.trans p-antitone p-shunting-swap regular-one-closed*)
qed
have $d^T * H * d \leq 1 \sqcup - H$
using *assms(1) big-forest-def dTransHd-le-1 le-supI1* **by** *blast*
then have $(- 1 \sqcap H) * d^T * H \leq - d^T$
by (*metis assms(1) big-forest-def dTransHd-le-1 inf.sup-monoid.add-commute le-infI2 p-antitone-iff regular-one-closed schroeder-4-p mult-assoc*)
then have $d * (- 1 \sqcap H) * d^T \leq - H$
by (*metis assms(1) big-forest-def conv-dist-comp schroeder-3-p triple-schroeder-p*)
then have $H \sqcap d * (- 1 \sqcap H) * d^T \leq 1$
by (*metis inf.coboundedI1 p-antitone-iff p-shunting-swap regular-one-closed*)
then have $H \sqcap d * d^T \sqcup H \sqcap d * (- 1 \sqcap H) * d^T \leq 1$

```
    using assms(1) big-forest-def le-supI by blast
  then have H ⊓ (d * 1 * d^T ⊔ d * (- 1 ⊓ H) * d^T) ≤ 1
    using comp-inf.semiring.distrib-left by auto
  then have H ⊓ (d * (1 ⊔ (- 1 ⊓ H)) * d^T) ≤ 1
    by (simp add: mult-left-dist-sup mult-right-dist-sup)
  then have 514: H ⊓ d * H * d^T ≤ 1
    by (metis assms(1) big-forest-def comp-inf.semiring.distrib-left inf.le-iff-sup
inf.sup-monoid.add-commute inf-top-right regular-one-closed stone)
  thus ?thesis
  proof -
    have H ⊓ d * H * d^T ⊔ H * d * H * (d * H)^⋆ ⊓ d * H * d^T ≤ 1
      using 513 514 by simp
    then have d * H * d^T ⊓ (H ⊔ H * d * H * (d * H)^⋆) ≤ 1
      by (simp add: comp-inf.semiring.distrib-left inf.sup-monoid.add-commute)
    then have d * H * d^T ⊓ H * (1 ⊔ d * H * (d * H)^⋆) ≤ 1
      by (simp add: mult-left-dist-sup mult-assoc)
    thus ?thesis
      by (simp add: inf.sup-monoid.add-commute star-left-unfold-equal)
  qed
qed
have ?x * top * ?x^T = (d ⊓ top * e^T * H ⊓ (H * d^T)^⋆ * H * a^T * top) * top * (d^T ⊓ H^T * e^TT
* top^T ⊓ top^T * a^TT * H^T * (d^TT * H^T)^⋆)
  by (simp add: conv-dist-comp conv-dist-inf conv-star-commute mult-assoc)
also have ... = (d ⊓ top * e^T * H ⊓ (H * d^T)^⋆ * H * a^T * top) * top * (d^T ⊓ H * e * top ⊓ top
* a * H * (d * H)^⋆)
  using assms(1) big-forest-def by auto
also have ... = (H * d^T)^⋆ * H * a^T * top ⊓ (d ⊓ top * e^T * H) * top * (d^T ⊓ H * e * top ⊓ top
* a * H * (d * H)^⋆)
  by (metis inf-vector-comp vector-export-comp)
also have ... = (H * d^T)^⋆ * H * a^T * top ⊓ (d ⊓ top * e^T * H) * top * top * (d^T ⊓ H * e * top
⊓ top * a * H * (d * H)^⋆)
  by (simp add: vector-mult-closed)
also have ... = (H * d^T)^⋆ * H * a^T * top ⊓ d * ((top * e^T * H)^T ⊓ top) * top * (d^T ⊓ H * e *
top ⊓ top * a * H * (d * H)^⋆)
  by (simp add: covector-comp-inf-1 covector-mult-closed)
also have ... = (H * d^T)^⋆ * H * a^T * top ⊓ d * ((top * e^T * H)^T ⊓ (H * e * top)^T) * d^T ⊓ top *
a * H * (d * H)^⋆
  by (smt comp-associative comp-inf.star-star-absorb comp-inf-vector conv-star-commute
covector-comp-inf covector-conv-vector fc-top star.circ-top total-conv-surjective vector-conv-covector
vector-inf-comp)
also have ... = (H * d^T)^⋆ * H * a^T * top ⊓ top * a * H * (d * H)^⋆ ⊓ d * ((top * e^T * H)^T ⊓ (H
* e * top)^T) * d^T
  using inf.sup-monoid.add-assoc inf.sup-monoid.add-commute by auto
also have ... = (H * d^T)^⋆ * H * a^T * top * top * a * H * (d * H)^⋆ ⊓ d * ((top * e^T * H)^T ⊓ (H
* e * top)^T) * d^T
  by (smt comp-inf.star.circ-decompose-9 comp-inf.star-star-absorb comp-inf-covector fc-top
star.circ-decompose-11 star.circ-top vector-export-comp)
also have ... = (H * d^T)^⋆ * H * a^T * top * a * H * (d * H)^⋆ ⊓ d * (H * e * top ⊓ top * e^T * H)
* d^T
  using assms(1) big-forest-def conv-dist-comp mult-assoc by auto
also have ... = (H * d^T)^⋆ * H * a^T * top * a * H * (d * H)^⋆ ⊓ d * H * e * top * e^T * H * d^T
  by (metis comp-inf-covector inf-top.left-neutral mult-assoc)
also have ... ≤ (H * d^T)^⋆ * (H * d)^⋆ * H ⊓ d * H * e * top * e^T * H * d^T
proof -
  have (H * d^T)^⋆ * H * a^T * top * a * H * (d * H)^⋆ ≤ (H * d^T)^⋆ * H * 1 * H * (d * H)^⋆
    using 2 comp-associative comp-isotone mult-left-isotone mult-semi-associative
star.circ-transitive-equal inf.sup-left-isotone by metis
  also have ... = (H * d^T)^⋆ * H * (d * H)^⋆
```

using *assms(1) big-forest-def mult.semigroup-axioms preorder-idempotent semigroup.assoc* **by** *fastforce*
also have $... = (H * d^T)^\star * (H * d)^\star * H$
by (*metis star-slide mult-assoc*)
finally show *?thesis*
using *inf.sup-left-isotone* **by** *auto*
qed
also have $... \leq (H * d^T)^\star * (H * d)^\star * H \sqcap d * H * d^T$
proof −
have $d * H * e * top * e^T * H * d^T \leq d * H * 1 * H * d^T$
using *3* **by** (*metis comp-isotone idempotent-one-closed mult-left-isotone mult-sub-right-one mult-assoc*)
also have $... \leq d * H * d^T$
by (*metis assms(1) big-forest-def mult-left-isotone mult-one-associative mult-semi-associative preorder-idempotent*)
finally show *?thesis*
using *inf.sup-right-isotone* **by** *auto*
qed
also have $... = H * (d^T * H)^\star * (H * d)^\star * H \sqcap d * H * d^T$
by (*metis assms(1) big-forest-def comp-associative preorder-idempotent star-slide*)
also have $... = H * ((d^T * H)^\star \sqcup (H * d)^\star) * H \sqcap d * H * d^T$
by (*simp add*: *assms(1) expand-big-forest mult.semigroup-axioms semigroup.assoc*)
also have $... = (H * (d^T * H)^\star * H \sqcup H * (H * d)^\star * H) \sqcap d * H * d^T$
by (*simp add*: *mult-left-dist-sup mult-right-dist-sup*)
also have $... = (H * d^T)^\star * H \sqcap d * H * d^T \sqcup H * (d * H)^\star \sqcap d * H * d^T$
by (*smt assms(1) big-forest-def inf-sup-distrib2 mult.semigroup-axioms preorder-idempotent star-slide semigroup.assoc*)
also have $... \leq (H * d^T)^\star * H \sqcap d * H * d^T \sqcup 1$
using *51 comp-inf.semiring.add-left-mono* **by** *blast*
finally have $?x * top * ?x^T \leq 1$
using *51 assms(1) big-forest-def conv-dist-comp conv-dist-inf conv-dist-sup conv-involutive conv-star-commute equivalence-one-closed mult.semigroup-axioms sup.absorb2 semigroup.assoc* **by** (*smt conv-isotone conv-order*)
thus *?thesis*
by *simp*
qed
have *6*: $?x^T * top * ?x \leq 1$
proof −
have $?x^T * top * ?x = (d^T \sqcap H^T * e^{TT} * top^T \sqcap top^T * a^{TT} * H^T * (d^{TT} * H^T)^\star) * top * (d \sqcap top * e^T * H \sqcap (H * d^T)^\star * H * a^T * top)$
by (*simp add*: *conv-dist-comp conv-dist-inf conv-star-commute mult-assoc*)
also have $... = (d^T \sqcap H * e * top \sqcap top * a * H * (d * H)^\star) * top * (d \sqcap top * e^T * H \sqcap (H * d^T)^\star * H * a^T * top)$
using *assms(1) big-forest-def* **by** *auto*
also have $... = H * e * top \sqcap (d^T \sqcap top * a * H * (d * H)^\star) * top * (d \sqcap top * e^T * H \sqcap (H * d^T)^\star * H * a^T * top)$
by (*smt comp-associative inf.sup-monoid.add-assoc inf.sup-monoid.add-commute star.circ-left-top star.circ-top vector-inf-comp*)
also have $... = H * e * top \sqcap d^T * ((top * a * H * (d * H)^\star)^T \sqcap top) * (d \sqcap top * e^T * H \sqcap (H * d^T)^\star * H * a^T * top)$
by (*simp add*: *covector-comp-inf-1 covector-mult-closed*)
also have $... = H * e * top \sqcap d^T * (d * H)^{\star T} * H * a^T * top * (d \sqcap top * e^T * H \sqcap (H * d^T)^\star * H * a^T * top)$
using *assms(1) big-forest-def comp-associative conv-dist-comp* **by** *auto*
also have $... = H * e * top \sqcap d^T * (d * H)^{\star T} * H * a^T * top * (d \sqcap (H * d^T)^\star * H * a^T * top) \sqcap top * e^T * H$
by (*smt comp-associative comp-inf-covector inf.sup-monoid.add-assoc inf.sup-monoid.add-commute*)

also have ... = $H * e * top \sqcap d^T * (d * H)^{\star T} * H * a^T * (top \sqcap ((H * d^T)^\star * H * a^T * top)^T) * d \sqcap top * e^T * H$
by (*metis comp-associative comp-inf-vector vector-conv-covector vector-top-closed*)
also have ... = $H * e * top \sqcap (H * e * top)^T \sqcap d^T * (d * H)^{\star T} * H * a^T * ((H * d^T)^\star * H * a^T * top)^T * d$
using *assms(1) big-forest-def conv-dist-comp inf.left-commute inf.sup-monoid.add-commute symmetric-top-closed mult-assoc* **by** (*smt inf-top.left-neutral*)
also have ... = $H * e * top * (H * e * top)^T \sqcap d^T * (d * H)^{\star T} * H * a^T * ((H * d^T)^\star * H * a^T * top)^T * d$
using *vector-covector vector-mult-closed* **by** *auto*
also have ... = $H * e * top * top^T * e^T * H^T \sqcap d^T * (d * H)^{\star T} * H * a^T * top^T * a^{TT} * H^T * (H * d^T)^{\star T} * d$
by (*smt conv-dist-comp mult.semigroup-axioms symmetric-top-closed semigroup.assoc*)
also have ... = $H * e * top * top * e^T * H \sqcap d^T * (H * d^T)^\star * H * a^T * top * a * H * (d * H)^\star * d$
using *assms(1) big-forest-def conv-dist-comp conv-star-commute* **by** *auto*
also have ... = $H * e * top * e^T * H \sqcap d^T * (H * d^T)^\star * H * a^T * top * a * H * (d * H)^\star * d$
using *vector-top-closed mult-assoc* **by** *auto*
also have ... $\leq H \sqcap d^T * (H * d^T)^\star * H * (d * H)^\star * d$
proof −
have $H * e * top * e^T * H \leq H * 1 * H$
using *3 comp-associative mult-left-isotone mult-right-isotone* **by** *metis*
also have ... = H
using *assms(1) big-forest-def preorder-idempotent* **by** *auto*
finally have *611*: $H * e * top * e^T * H \leq H$
by *simp*
have $d^T * (H * d^T)^\star * H * a^T * top * a * H * (d * H)^\star * d \leq d^T * (H * d^T)^\star * H * 1 * H * (d * H)^\star * d$
using *2 comp-associative mult-left-isotone mult-right-isotone* **by** *metis*
also have ... = $d^T * (H * d^T)^\star * H * (d * H)^\star * d$
using *assms(1) big-forest-def mult.semigroup-axioms preorder-idempotent semigroup.assoc* **by** *fastforce*
finally have $d^T * (H * d^T)^\star * H * a^T * top * a * H * (d * H)^\star * d \leq d^T * (H * d^T)^\star * H * (d * H)^\star * d$
by *simp*
thus *?thesis*
using *611 comp-inf.comp-isotone* **by** *blast*
qed
also have ... = $H \sqcap (d^T * H)^\star * d^T * H * d * (H * d)^\star$
using *star-slide mult-assoc* **by** *auto*
also have ... $\leq H \sqcap (d^T * H)^\star * (H * d)^\star$
proof −
have $(d^T * H)^\star * d^T * H * d * (H * d)^\star \leq (d^T * H)^\star * 1 * (H * d)^\star$
by (*smt assms(1) big-forest-def conv-dist-comp mult-left-isotone mult-right-isotone preorder-idempotent mult-assoc*)
also have ... = $(d^T * H)^\star * (H * d)^\star$
by *simp*
finally show *?thesis*
using *inf.sup-right-isotone* **by** *blast*
qed
also have ... = $H \sqcap ((d^T * H)^\star \sqcup (H * d)^\star)$
by (*simp add*: *assms(1) expand-big-forest*)
also have ... = $H \sqcap (d^T * H)^\star \sqcup H \sqcap (H * d)^\star$
by (*simp add*: *comp-inf.semiring.distrib-left*)
also have ... = $1 \sqcup H \sqcap (d^T * H)^+ \sqcup H \sqcap (H * d)^+$
proof −
have *612*: $H \sqcap (H * d)^\star = 1 \sqcup H \sqcap (H * d)^+$
using *assms(1) big-forest-def reflexive-inf-star* **by** *blast*

```
    have H ⊓ (d^T * H)^⋆ = 1 ⊔ H ⊓ (d^T * H)^+
      using assms(1) big-forest-def reflexive-inf-star by auto
    thus ?thesis
      using 612 sup-assoc sup-commute by auto
  qed
  also have ... ≤ 1
  proof −
    have 613: H ⊓ (H * d)^+ ≤ 1
      by (metis assms(3) inf.coboundedI1 p-antitone-iff p-shunting-swap regular-one-closed)
    then have H ⊓ (d^T * H)^+ ≤ 1
      by (metis assms(1) big-forest-def conv-dist-comp conv-dist-inf conv-plus-commute
coreflexive-symmetric)
    thus ?thesis
      by (simp add: 613)
  qed
  finally show ?thesis
    by simp
 qed
 have 7:bijective (?x * top)
  using 4 5 6 arc-expanded by blast
 have bijective (?x^T * top)
  using 4 5 6 arc-expanded by blast
 thus ?thesis
  using 7 by simp
qed
```

Theorem 8

```
lemma e-leq-c-c-complement-transpose-general:
 assumes e = minarc (c * −(c)^T ⊓ g)
  and regular c
 shows e ≤ c * −(c)^T
proof −
 have e ≤ −− (c * − c^T ⊓ g)
  using assms(1) minarc-below order-trans by blast
 also have ... ≤ −− (c * − c^T)
  using order-lesseq-imp pp-isotone-inf by blast
 also have ... = c * − c^T
  using assms(2) regular-mult-closed by auto
 finally show ?thesis
  by simp
qed
```

Theorem 9

```
lemma x-leq-c-transpose-general:
 assumes forest h
  and vector c
  and x^T * top ≤ forest-components(h) * e * top
  and e ≤ c * −c^T
  and c = forest-components(h) * c
 shows x ≤ c^T
proof −
 let ?H = forest-components h
 have x ≤ top * x
  using top-left-mult-increasing by blast
 also have ... ≤ (?H * e * top)^T
  using assms(3) conv-dist-comp conv-order by force
 also have ... = top * e^T * ?H
  using assms(1) comp-associative conv-dist-comp forest-components-equivalence by auto
```

```
  also have ... ≤ top * (c * - c^T)^T * ?H
    by (simp add: assms(4) conv-isotone mult-left-isotone mult-right-isotone)
  also have ... = top * (- c * c^T) * ?H
    by (simp add: conv-complement conv-dist-comp)
  also have ... ≤ top * c^T * ?H
    by (metis mult-left-isotone top.extremum mult-assoc)
  also have ... = c^T * ?H
    using assms(1, 2) component-is-vector vector-conv-covector by auto
  also have ... = c^T
    by (metis assms(1) assms(5) fch-equivalence conv-dist-comp)
  finally show ?thesis
    by simp
qed
```

Theorem 10

```
lemma x-leq-c-complement-general:
  assumes vector c
    and c * c^T ≤ forest-components h
    and x ≤ c^T
    and x ≤ -forest-components h
  shows x ≤ -c
proof -
  let ?H = forest-components h
  have x ≤ - ?H ⊓ c^T
    using assms(3, 4) by auto
  also have ... ≤ - c
  proof -
    have c ⊓ c^T ≤ ?H
      using assms(1, 2) vector-covector by auto
    then have -?H ⊓ c ⊓ c^T ≤ bot
      using inf.sup-monoid.add-assoc p-antitone pseudo-complement by fastforce
    thus ?thesis
      using le-bot p-shunting-swap pseudo-complement by blast
  qed
  finally show ?thesis
    by simp
qed
```

Theorem 11

```
lemma sum-e-below-sum-x-when-outgoing-same-component-general:
  assumes e = minarc (c * -(c)^T ⊓ g)
    and regular c
    and forest h
    and vector c
    and x^T * top ≤ (forest-components h) * e * top
    and c = (forest-components h) * c
    and c * c^T ≤ forest-components h
    and x ≤ - forest-components h ⊓ -- g
    and symmetric g
    and arc x
    and c ≠ bot
  shows sum (e ⊓ g) ≤ sum (x ⊓ g)
proof -
  let ?H = forest-components h
  have 1:e ≤ c * - c^T
    using assms(1, 2) e-leq-c-c-complement-transpose-general by auto
  have 2: x ≤ c^T
    using 1 assms(3, 4, 5, 6) x-leq-c-transpose-general by auto
```

```
  then have x ≤ -c
    using assms(4, 7, 8) x-leq-c-complement-general inf.boundedE by blast
  then have x ≤ - c ⊓ c^T
    using 2 by simp
  then have x ≤ - c * c^T
    using assms(4) by (simp add: vector-complement-closed vector-covector)
  then have x^T ≤ c^TT * - c^T
    by (metis conv-complement conv-dist-comp conv-isotone)
  then have 3: x^T ≤ c * - c^T
    by simp
  then have x ≤ -- g
    using assms(8) by auto
  then have x^T ≤ -- g
    using assms(9) conv-complement conv-isotone by fastforce
  then have x^T ⊓ c * - c^T ⊓ -- g ≠ bot
    using 3 assms(10, 11) by (metis comp-inf.semiring.mult-not-zero conv-dist-comp
        conv-involutive inf.orderE mult-right-zero top.extremum)
  then have x^T ⊓ c * - c^T ⊓ g ≠ bot
    using inf.sup-monoid.add-commute pp-inf-bot-iff by auto
  then have sum (minarc (c * - c^T ⊓ g) ⊓ (c * - c^T ⊓ g)) ≤ sum (x^T ⊓ c * - c^T ⊓ g)
    using assms(10) minarc-min inf.sup-monoid.add-assoc by auto
  then have sum (e ⊓ c * - c^T ⊓ g) ≤ sum (x^T ⊓ c * - c^T ⊓ g)
    using assms(1) inf.sup-monoid.add-assoc by auto
  then have sum (e ⊓ g) ≤ sum (x^T ⊓ g)
    using 1 3 inf.orderE by metis
  then have sum (e ⊓ g) ≤ sum (x ⊓ g)
    using assms(9) sum-symmetric by auto
  thus ?thesis
    by simp
qed

lemma sum-e-below-sum-x-when-outgoing-same-component:
  assumes symmetric g
    and vector j
    and forest h
    and x ≤ - forest-components h ⊓ -- g
    and x^T * top ≤ forest-components h * selected-edge h j g * top
    and j ≠ bot
    and arc x
  shows sum (selected-edge h j g ⊓ g) ≤ sum (x ⊓ g)
proof -
  let ?e = selected-edge h j g
  let ?c = choose-component (forest-components h) j
  let ?H = forest-components h
  show ?thesis
  proof (rule sum-e-below-sum-x-when-outgoing-same-component-general)
  next
    show ?e = minarc (?c * - ?c^T ⊓ g)
      by simp
  next
    show regular ?c
      using component-is-regular by auto
  next
    show forest h
      by (simp add: assms(3))
  next
    show vector ?c
      by (simp add: assms(2, 6) component-is-vector)
```

```
  next
    show x^T * top  ≤ ?H * ?e * top
      by (simp add: assms(5))
  next
    show ?c = ?H * ?c
      using component-single by auto
  next
    show ?c * ?c^T ≤ ?H
      by (simp add: component-is-connected)
  next
    show x ≤ -?H ⊓ -- g
      using assms(4) by auto
  next
    show symmetric g
      by (simp add: assms(1))
  next
    show arc x
      by (simp add: assms(7))
  next
    show ?c ≠ bot
      using assms(2, 5 , 6, 7) inf-bot-left le-bot minarc-bot mult-left-zero mult-right-zero by fastforce
  qed
qed

lemma a-to-e-in-bigforest:
  assumes symmetric g
    and f ≤ --g
    and vector j
    and forest h
    and big-forest (forest-components h) d
    and f ⊔ f^T = h ⊔ h^T ⊔ d ⊔ d^T
    and (∀ a b . bf-between-arcs a b (forest-components h) d ∧ a ≤ -(forest-components h) ⊓ -- g ∧
b ≤ d ⟶ sum(b ⊓ g) ≤ sum(a ⊓ g))
    and regular d
    and j ≠ bot
    and b = selected-edge h j g
    and arc a
    and bf-between-arcs a b (forest-components h) (d ⊔ selected-edge h j g)
    and a ≤ - forest-components h ⊓ -- g
    and regular h
  shows sum (b ⊓ g) ≤ sum (a ⊓ g)
proof -
  let ?p = path f h j g
  let ?e = selected-edge h j g
  let ?F = forest-components f
  let ?H = forest-components h
  have sum (b ⊓ g) ≤ sum (a ⊓ g)
  proof (cases a^T * top ≤ ?H * ?e * top)
    case True
    show a^T * top ≤ ?H * ?e * top ⟹ sum (b ⊓ g) ≤ sum (a ⊓ g)
    proof-
      have sum (?e ⊓ g) ≤ sum (a ⊓ g)
      proof (rule sum-e-below-sum-x-when-outgoing-same-component)
        show symmetric g
          using assms(1) by auto
      next
        show vector j
          using assms(3) by blast
```

```
    next
      show forest h
        by (simp add: assms(4))
    next
      show a ≤ - ?H ⊓ -- g
        using assms(13) by auto
    next
      show a^T * top ≤ ?H * ?e * top
        using True by auto
    next
      show j ≠ bot
        by (simp add: assms(9))
    next
      show arc a
        by (simp add: assms(11))
    qed
    thus ?thesis
      using assms(10) by auto
  qed
next
  case False
  show ¬ a^T * top ≤ ?H * ?e * top ⟹ sum (b ⊓ g) ≤ sum (a ⊓ g)
  proof -
    let ?d' = d ⊔ ?e
    let ?x = d ⊓ top * ?e^T * ?H ⊓ (?H * d^T)^⋆ * ?H * a^T * top
    have 61: arc (?x)
    proof (rule shows-arc-x)
      show big-forest ?H d
        by (simp add: assms(5))
    next
      show bf-between-arcs a ?e ?H d
      proof -
        have 611: bf-between-arcs a b ?H (d ⊔ b)
          using assms(10) assms(12) by auto
        have 616: regular h
          using assms(14) by auto
        have regular a
          using 611 bf-between-arcs-def arc-regular by fastforce
        thus ?thesis
          by (smt 616 big-forest-path-split-disj assms(4, 8, 10, 12) bf-between-arcs-def fch-equivalence
minarc-regular regular-closed-star regular-conv-closed regular-mult-closed)
      qed
    next
      show (?H * d)^+ ≤ - ?H
        using assms(5) big-forest-def by blast
    next
      show ¬ a^T * top ≤ ?H * ?e * top
        by (simp add: False)
    next
      show regular a
        using assms(12) bf-between-arcs-def arc-regular by auto
    next
      show regular ?e
        using minarc-regular by auto
    next
      show regular ?H
        using assms(14) pp-dist-star regular-conv-closed regular-mult-closed by auto
    next
```

```
  show regular d
    using assms(8) by auto
qed
have 62: bijective (a^T * top)
  by (simp add: assms(11))
have 63: bijective (?x * top)
  using 61 by simp
have 64: ?x ≤ (?H * d^T)^⋆ * ?H * a^T * top
  by simp
then have ?x * top ≤ (?H * d^T)^⋆ * ?H * a^T * top
  using mult-left-isotone inf-vector-comp by auto
then have a^T * top ≤ ((?H * d^T)^⋆ * ?H)^T * ?x * top
  using 62 63 64 bijective-reverse mult-assoc by smt
also have ... = ?H * (d * ?H)^⋆ * ?x * top
  using conv-dist-comp conv-star-commute by auto
also have ... = (?H * d)^⋆ * ?H * ?x * top
  by (simp add: star-slide)
finally have a^T * top ≤ (?H * d)^⋆ * ?H * ?x * top
  by simp
then have 65: bf-between-arcs a ?x ?H d
  using 61 assms(12) bf-between-arcs-def by blast
have 66: ?x ≤ d
  by (simp add: inf.sup-monoid.add-assoc)
then have x-below-a: sum (?x ⊓ g) ≤ sum (a ⊓ g)
  using 65 bf-between-arcs-def assms(7) assms(13) by blast
have sum (?e ⊓ g) ≤ sum (?x ⊓ g)
proof (rule sum-e-below-sum-x-when-outgoing-same-component)
  show symmetric g
    using assms(1) by auto
next
  show vector j
    using assms(3) by blast
next
  show forest h
    by (simp add: assms(4))
next
  show ?x ≤ - ?H ⊓ -- g
  proof -
    have 67: ?x ≤ - ?H
      using 66 assms(5) big-forest-def order-lesseq-imp by blast
    have ?x ≤ d
      by (simp add: conv-isotone inf.sup-monoid.add-assoc)
    also have ... ≤ f ⊔ f^T
    proof -
      have h ⊔ h^T ⊔ d ⊔ d^T = f ⊔ f^T
        by (simp add: assms(6))
      then show ?thesis
        by (metis (no-types) le-supE sup.absorb-iff2 sup.idem)
    qed
    also have ... ≤ -- g
      using assms(1) assms(2) conv-complement conv-isotone by fastforce
    finally have ?x ≤ -- g
      by simp
    thus ?thesis
      by (simp add: 67)
  qed
next
  show ?x^T * top ≤ ?H * ?e * top
```

```
      proof −
        have ?x ≤ top ∗ ?e^T ∗ ?H
          using inf.coboundedI1 by auto
        then have ?x^T ≤ ?H ∗ ?e ∗ top
          using conv-dist-comp conv-dist-inf conv-star-commute inf.orderI inf.sup-monoid.add-assoc
inf.sup-monoid.add-commute mult-assoc by auto
        then have ?x^T ∗ top ≤ ?H ∗ ?e ∗ top ∗ top
          by (simp add: mult-left-isotone)
        thus ?thesis
          by (simp add: mult-assoc)
      qed
    next
      show j ≠ bot
        by (simp add: assms(9))
    next
      show arc (?x)
        using 61 by blast
    qed
    then have sum (?e ⊓ g) ≤ sum (a ⊓ g)
      using x-below-a order.trans by blast
    thus ?thesis
      by (simp add: assms(10))
  qed
 qed
 thus ?thesis
  by simp
qed

lemma boruvka-exchange-spanning-inv:
 assumes forest v
  and v^⋆ ∗ e^T = e^T
  and i ≤ v ⊓ top ∗ e^T ∗ w^{T⋆}
  and arc i
  and arc e
  and v ≤ −−g
  and w ≤ −−g
  and e ≤ −−g
  and components g ≤ forest-components v
 shows i ≤ (v ⊓ −i)^{T⋆} ∗ e^T ∗ top
proof −
 have 1: (v ⊓ −i ⊓ −i^T) ∗ (v^T ⊓ −i ⊓ −i^T) ≤ 1
  using assms(1) comp-isotone order.trans inf.cobounded1 by blast
 have 2: bijective (i ∗ top) ∧ bijective (e^T ∗ top)
  using assms(4, 5) mult-assoc by auto
 have i ≤ v ∗ (top ∗ e^T ∗ w^{T⋆})^T
  using assms(3) covector-mult-closed covector-restrict-comp-conv order-lesseq-imp vector-top-closed
by blast
 also have ... ≤ v ∗ w^{T⋆T} ∗ e^{TT} ∗ top^T
  by (simp add: comp-associative conv-dist-comp)
 also have ... ≤ v ∗ w^⋆ ∗ e ∗ top
  by (simp add: conv-star-commute)
 also have ... = v ∗ w^⋆ ∗ e ∗ e^T ∗ e ∗ top
  using assms(5) arc-eq-1 by (simp add: comp-associative)
 also have ... ≤ v ∗ w^⋆ ∗ e ∗ e^T ∗ top
  by (simp add: comp-associative mult-right-isotone)
 also have ... ≤ (−−g) ∗ (−−g)^⋆ ∗ (−−g) ∗ e^T ∗ top
  using assms(6, 7, 8) by (simp add: comp-isotone star-isotone)
 also have ... ≤ (−−g)^⋆ ∗ e^T ∗ top
```

by (*metis comp-isotone mult-left-isotone star.circ-increasing star.circ-transitive-equal*)
also have ... $\leq v^{T\star} * v^\star * e^T * top$
by (*simp add*: *assms*(*9*) *mult-left-isotone*)
also have ... $\leq v^{T\star} * e^T * top$
by (*simp add*: *assms*(*2*) *comp-associative*)
finally have $i \leq v^{T\star} * e^T * top$
by *simp*
then have $i * top \leq v^{T\star} * e^T * top$
by (*metis comp-associative mult-left-isotone vector-top-closed*)
then have $e^T * top \leq v^{T\star T} * i * top$
using *2 bijective-reverse mult-assoc* **by** *metis*
also have ... $= v^\star * i * top$
by (*simp add*: *conv-star-commute*)
also have ... $\leq (v \sqcap -i \sqcap -i^T)^\star * i * top$
proof −
have *3*: $i * top \leq (v \sqcap -i \sqcap -i^T)^\star * i * top$
using *star.circ-loop-fixpoint sup-right-divisibility mult-assoc* **by** *auto*
have $(v \sqcap i) * (v \sqcap -i \sqcap -i^T)^\star * i * top \leq i * top * i * top$
using *comp-isotone inf.cobounded1 inf.sup-monoid.add-commute mult-left-isotone top.extremum*
by *presburger*
also have ... $\leq i * top$
by *simp*
finally have *4*: $(v \sqcap i) * (v \sqcap -i \sqcap -i^T)^\star * i * top \leq (v \sqcap -i \sqcap -i^T)^\star * i * top$
using *3 dual-order.trans* **by** *blast*
have *5*: $(v \sqcap -i \sqcap -i^T) * (v \sqcap -i \sqcap -i^T)^\star * i * top \leq (v \sqcap -i \sqcap -i^T)^\star * i * top$
by (*metis mult-left-isotone star.circ-increasing star.left-plus-circ*)
have $v^+ \leq -1$
by (*simp add*: *assms*(*1*))
then have $v * v \leq -1$
by (*metis mult-left-isotone order-trans star.circ-increasing star.circ-plus-same*)
then have $v * 1 \leq -v^T$
by (*simp add*: *schroeder-5-p*)
then have $v \leq -v^T$
by *simp*
then have $v \sqcap v^T \leq bot$
by (*simp add*: *bot-unique pseudo-complement*)
then have *7*: $v \sqcap i^T \leq bot$
by (*metis assms*(*3*) *comp-inf.mult-right-isotone conv-dist-inf inf.boundedE inf.le-iff-sup le-bot*)
then have $(v \sqcap i^T) * (v \sqcap -i \sqcap -i^T)^\star * i * top \leq bot$
using *le-bot semiring.mult-zero-left* **by** *fastforce*
then have *6*: $(v \sqcap i^T) * (v \sqcap -i \sqcap -i^T)^\star * i * top \leq (v \sqcap -i \sqcap -i^T)^\star * i * top$
using *bot-least le-bot* **by** *blast*
have *8*: $v = (v \sqcap i) \sqcup (v \sqcap i^T) \sqcup (v \sqcap -i \sqcap -i^T)$
proof −
have *81*: *regular i*
by (*simp add*: *assms*(*4*) *arc-regular*)
have $(v \sqcap i^T) \sqcup (v \sqcap -i \sqcap -i^T) = (v \sqcap -i)$
using *7* **by** (*metis comp-inf.coreflexive-comp-inf-complement inf-import-p inf-p le-bot*
maddux-3-11-pp top.extremum)
then have $(v \sqcap i) \sqcup (v \sqcap i^T) \sqcup (v \sqcap -i \sqcap -i^T) = (v \sqcap i) \sqcup (v \sqcap -i)$
by (*simp add*: *sup.semigroup-axioms semigroup.assoc*)
also have ... $= v$
using *81* **by** (*metis maddux-3-11-pp*)
finally show *?thesis*
by *simp*
qed
have $(v \sqcap i) * (v \sqcap -i \sqcap -i^T)^\star * i * top \sqcup (v \sqcap i^T) * (v \sqcap -i \sqcap -i^T)^\star * i * top \sqcup (v \sqcap -i \sqcap$
$-i^T) * (v \sqcap -i \sqcap -i^T)^\star * i * top \leq (v \sqcap -i \sqcap -i^T)^\star * i * top$

```
    using 4 5 6 by simp
  then have ((v ⊓ i) ⊔ (v ⊓ i^T) ⊔ (v ⊓ -i ⊓ -i^T)) * (v ⊓ -i ⊓ -i^T)^⋆ * i * top ≤ (v ⊓ -i ⊓
-i^T)^⋆ * i * top
    by (simp add: mult-right-dist-sup)
  then have v * (v ⊓ -i ⊓ -i^T)^⋆ * i * top ≤ (v ⊓ -i ⊓ -i^T)^⋆ * i * top
    using 8 by auto
  then have i * top ⊔ v * (v ⊓ -i ⊓ -i^T)^⋆ * i * top ≤ (v ⊓ -i ⊓ -i^T)^⋆ * i * top
    using 3 by auto
  then have 9:v^⋆ * (v ⊓ -i ⊓ -i^T)^⋆ * i * top ≤ (v ⊓ -i ⊓ -i^T)^⋆ * i * top
    by (simp add: star-left-induct-mult mult-assoc)
  have v^⋆ * i * top ≤ v^⋆ * (v ⊓ -i ⊓ -i^T)^⋆ * i * top
    using 3 mult-right-isotone mult-assoc by auto
  thus ?thesis
    using 9 order.trans by blast
 qed
 finally have e^T * top ≤ (v ⊓ -i ⊓ -i^T)^⋆ * i * top
  by simp
 then have i * top ≤ (v ⊓ -i ⊓ -i^T)^⋆T * e^T * top
  using 2 bijective-reverse mult-assoc by metis
 also have ... = (v^T ⊓ -i ⊓ -i^T)^⋆ * e^T * top
  using comp-inf.inf-vector-comp conv-complement conv-dist-inf conv-star-commute
inf.sup-monoid.add-commute by auto
 also have ... ≤ ((v ⊓ -i ⊓ -i^T) ⊔ (v^T ⊓ -i ⊓ -i^T))^⋆ * e^T * top
  by (simp add: mult-left-isotone star-isotone)
 finally have i ≤ ((v^T ⊓ -i ⊓ -i^T) ⊔ (v ⊓ -i ⊓ -i^T))^⋆ * e^T * top
  using dual-order.trans top-right-mult-increasing sup-commute by presburger
 also have ... = (v^T ⊓ -i ⊓ -i^T)^⋆ * (v ⊓ -i ⊓ -i^T)^⋆ * e^T * top
  using 1 cancel-separate-1 by (simp add: sup-commute)
 also have ... ≤ (v^T ⊓ -i ⊓ -i^T)^⋆ * v^⋆ * e^T * top
  by (simp add: inf-assoc mult-left-isotone mult-right-isotone star-isotone)
 also have ... = (v^T ⊓ -i ⊓ -i^T)^⋆ * e^T * top
  using assms(2) mult-assoc by simp
 also have ... ≤ (v^T ⊓ -i^T)^⋆ * e^T * top
  using mult-left-isotone conv-isotone star-isotone comp-inf.mult-right-isotone inf.cobounded2
inf.left-commute inf.sup-monoid.add-commute by presburger
 also have ... = (v ⊓ -i)^T⋆ * e^T * top
  using conv-complement conv-dist-inf by presburger
 finally show ?thesis
  by simp
qed

lemma boruvka-edge-arc:
 assumes equivalence F
  and forest v
  and arc e
  and regular F
  and F ≤ forest-components (F ⊓ v)
  and regular v
  and v * e^T = bot
  and e * F * e = bot
  and e^T ≤ v^⋆
  and e ≠ bot
 shows arc (v ⊓ -F * e * top ⊓ top * e^T * F)
proof -
 let ?i = v ⊓ -F * e * top ⊓ top * e^T * F
 have 1: ?i^T * top * ?i ≤ 1
 proof -
  have ?i^T * top * ?i = (v^T ⊓ top * e^T * -F ⊓ F * e * top) * top * (v ⊓ -F * e * top ⊓ top * e^T
```

$* F$)
using *assms(1) conv-complement conv-dist-comp conv-dist-inf mult.semigroup-axioms semigroup.assoc* **by** *fastforce*
also have ... $= F * e * top \sqcap (v^T \sqcap top * e^T * -F) * top * (v \sqcap -F * e * top) \sqcap top * e^T * F$
by (*smt covector-comp-inf covector-mult-closed inf-vector-comp vector-export-comp vector-top-closed*)
also have ... $= F * e * top \sqcap (v^T \sqcap top * e^T * -F) * top * top * (v \sqcap -F * e * top) \sqcap top * e^T * F$
by (*simp add*: *comp-associative*)
also have ... $= F * e * top \sqcap v^T * (top \sqcap (top * e^T * -F)^T) * top * (v \sqcap -F * e * top) \sqcap top * e^T * F$
using *comp-associative comp-inf-vector-1* **by** *auto*
also have ... $= F * e * top \sqcap v^T * (top \sqcap (top * e^T * -F)^T) * (top \sqcap (-F * e * top)^T) * v \sqcap top * e^T * F$
by (*smt comp-inf-vector conv-dist-comp mult.semigroup-axioms symmetric-top-closed semigroup.assoc*)
also have ... $= F * e * top \sqcap v^T * (top * e^T * -F)^T * (-F * e * top)^T * v \sqcap top * e^T * F$
by *simp*
also have ... $= F * e * top \sqcap v^T * -F^T * e^{TT} * top^T * top^T * e^T * -F^T * v \sqcap top * e^T * F$
using *comp-associative conv-complement conv-dist-comp* **by** *presburger*
also have ... $= F * e * top \sqcap v^T * -F * e * top * top * e^T * -F * v \sqcap top * e^T * F$
by (*simp add*: *assms(1)*)
also have ... $= F * e * top \sqcap v^T * -F * e * top \sqcap top * e^T * -F * v \sqcap top * e^T * F$
by (*metis comp-associative comp-inf-covector inf.sup-monoid.add-assoc inf-top.left-neutral vector-top-closed*)
also have ... $= (F \sqcap v^T * -F) * e * top \sqcap top * e^T * -F * v \sqcap top * e^T * F$
using *assms(3) injective-comp-right-dist-inf mult-assoc* **by** *auto*
also have ... $= (F \sqcap v^T * -F) * e * top \sqcap top * e^T * (F \sqcap -F * v)$
using *assms(3) conv-dist-comp inf.sup-monoid.add-assoc inf.sup-monoid.add-commute mult.semigroup-axioms univalent-comp-left-dist-inf semigroup.assoc* **by** *fastforce*
also have ... $= (F \sqcap v^T * -F) * e * top * top * e^T * (F \sqcap -F * v)$
by (*metis comp-associative comp-inf-covector inf-top.left-neutral vector-top-closed*)
also have ... $= (F \sqcap v^T * -F) * e * top * e^T * (F \sqcap -F * v)$
by (*simp add*: *comp-associative*)
also have ... $\leq (F \sqcap v^T * -F) * (F \sqcap -F * v)$
using *assms(3)* **by** (*smt conv-dist-comp mult-left-isotone shunt-bijective symmetric-top-closed top-right-mult-increasing mult-assoc*)
also have ... $\leq (F \sqcap v^T * -F) * (F \sqcap -F * v) \sqcap F$
by (*metis assms(1) inf.absorb1 inf.cobounded1 mult-isotone preorder-idempotent*)
also have ... $\leq (F \sqcap v^T * -F) * (F \sqcap -F * v) \sqcap (F \sqcap v)^{T\star} * (F \sqcap v)^\star$
using *assms(5) comp-inf.mult-right-isotone* **by** *auto*
also have ... $\leq (-F \sqcap v^T) * -F * -F * (-F \sqcap v) \sqcap (F \sqcap v)^{T\star} * (F \sqcap v)^\star$
proof −
have $F \sqcap v^T * -F \leq (v^T \sqcap F * -F^T) * -F$
by (*metis conv-complement dedekind-2 inf-commute*)
also have ... $= (v^T \sqcap -F^T) * -F$
using *assms(1) equivalence-comp-left-complement* **by** *simp*
finally have $F \sqcap v^T * -F \leq F \sqcap (v^T \sqcap -F) * -F$
using *assms(1)* **by** *auto*
then have *11*: $F \sqcap v^T * -F = F \sqcap (-F \sqcap v^T) * -F$
using *assms(1) inf.antisym-conv inf.sup-monoid.add-commute* **by** (*metis comp-left-subdist-inf inf.boundedE inf.sup-right-isotone*)
then have $F^T \sqcap -F^T * v^{TT} = F^T \sqcap -F^T * (-F^T \sqcap v^{TT})$
by (*metis* (*full-types*) *assms(1) conv-complement conv-dist-comp conv-dist-inf*)
then have *12*: $F \sqcap -F * v = F \sqcap -F * (-F \sqcap v)$
using *assms(1)* **by** (*simp add*: *abel-semigroup.commute inf.abel-semigroup-axioms*)
have $(F \sqcap v^T * -F) * (F \sqcap -F * v) = (F \sqcap (-F \sqcap v^T) * -F) * (F \sqcap -F * (-F \sqcap v))$
using *11 12* **by** *auto*

```
    also have ... ≤ (−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v)
      by (metis comp-associative comp-isotone inf.cobounded2)
    finally show ?thesis
      using comp-inf.mult-left-isotone by blast
  qed
  also have ... = ((−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ (F ⊓ v)^T ∗ (F ⊓ v)^T⋆ ∗ (F ⊓ v)⋆) ⊔ ((−F
⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ (F ⊓ v)⋆)
    by (metis comp-associative inf-sup-distrib1 star.circ-loop-fixpoint)
  also have ... = ((−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ (F ⊓ v^T) ∗ (F ⊓ v)^T⋆ ∗ (F ⊓ v)⋆) ⊔ ((−F
⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ (F ⊓ v)⋆)
    using assms(1) conv-dist-inf by auto
  also have ... = bot ⊔ ((−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ (F ⊓ v)⋆)
  proof −
    have (−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ (F ⊓ v^T) ∗ (F ⊓ v)^T⋆ ∗ (F ⊓ v)⋆ ≤ bot
      using assms(1, 2) forests-bot-2 by (simp add: comp-associative)
    thus ?thesis
      using le-bot by blast
  qed
  also have ... = (−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ (1 ⊔ (F ⊓ v)⋆ ∗ (F ⊓ v))
    by (simp add: star.circ-plus-same star-left-unfold-equal)
  also have ... = ((−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ 1) ⊔ ((−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v)
⊓ (F ⊓ v)⋆ ∗ (F ⊓ v))
    by (simp add: comp-inf.semiring.distrib-left)
  also have ... ≤ 1 ⊔ ((−F ⊓ v^T) ∗ −F ∗ −F ∗ (−F ⊓ v) ⊓ (F ⊓ v)⋆ ∗ (F ⊓ v))
    using sup-left-isotone by auto
  also have ... ≤ 1 ⊔ bot
    using assms(1, 2) forests-bot-3 comp-inf.semiring.add-left-mono by simp
  finally show ?thesis
    by simp
qed
have 2: ?i ∗ top ∗ ?i^T ≤ 1
proof −
  have ?i ∗ top ∗ ?i^T = (v ⊓ −F ∗ e ∗ top ⊓ top ∗ e^T ∗ F) ∗ top ∗ (v^T ⊓ (−F ∗ e ∗ top)^T ⊓ (top ∗
e^T ∗ F)^T)
    by (simp add: conv-dist-inf)
  also have ... = (v ⊓ −F ∗ e ∗ top ⊓ top ∗ e^T ∗ F) ∗ top ∗ (v^T ⊓ top^T ∗ e^T ∗ −F^T ⊓ F^T ∗ e^TT ∗
top^T)
    by (simp add: conv-complement conv-dist-comp mult-assoc)
  also have ... = (v ⊓ −F ∗ e ∗ top ⊓ top ∗ e^T ∗ F) ∗ top ∗ (v^T ⊓ top ∗ e^T ∗ −F ⊓ F ∗ e ∗ top)
    by (simp add: assms(1))
  also have ... = −F ∗ e ∗ top ⊓ (v ⊓ top ∗ e^T ∗ F) ∗ top ∗ (v^T ⊓ top ∗ e^T ∗ −F ⊓ F ∗ e ∗ top)
    by (smt inf.left-commute inf.sup-monoid.add-assoc vector-export-comp)
  also have ... = −F ∗ e ∗ top ⊓ (v ⊓ top ∗ e^T ∗ F) ∗ top ∗ (v^T ⊓ F ∗ e ∗ top) ⊓ top ∗ e^T ∗ −F
    by (smt comp-inf-covector inf.sup-monoid.add-assoc inf.sup-monoid.add-commute mult-assoc)
  also have ... = −F ∗ e ∗ top ⊓ (v ⊓ top ∗ e^T ∗ F) ∗ top ∗ top ∗ (v^T ⊓ F ∗ e ∗ top) ⊓ top ∗ e^T ∗
−F
    by (simp add: mult-assoc)
  also have ... = −F ∗ e ∗ top ⊓ v ∗ ((top ∗ e^T ∗ F)^T ⊓ top) ∗ top ∗ (v^T ⊓ F ∗ e ∗ top) ⊓ top ∗ e^T
∗ −F
    by (simp add: comp-inf-vector-1 mult.semigroup-axioms semigroup.assoc)
  also have ... = −F ∗ e ∗ top ⊓ v ∗ ((top ∗ e^T ∗ F)^T ⊓ top) ∗ (top ⊓ (F ∗ e ∗ top)^T) ∗ v^T ⊓ top ∗
e^T ∗ −F
    by (smt comp-inf-vector covector-comp-inf vector-conv-covector vector-mult-closed
vector-top-closed)
  also have ... = −F ∗ e ∗ top ⊓ v ∗ (top ∗ e^T ∗ F)^T ∗ (F ∗ e ∗ top)^T ∗ v^T ⊓ top ∗ e^T ∗ −F
    by simp
  also have ... = −F ∗ e ∗ top ⊓ v ∗ F^T ∗ e^TT ∗ top^T ∗ top^T ∗ e^T ∗ F^T ∗ v^T ⊓ top ∗ e^T ∗ −F
    using comp-associative conv-dist-comp by presburger
```

also have ... = $-F * e * top \sqcap v * F * e * top * top * e^T * F * v^T \sqcap top * e^T * -F$
using *assms(1)* **by** *auto*
also have ... = $-F * e * top \sqcap v * F * e * top \sqcap top * e^T * F * v^T \sqcap top * e^T * -F$
by (*smt comp-associative comp-inf-covector inf.sup-monoid.add-assoc inf-top.left-neutral vector-top-closed*)
also have ... = $(-F \sqcap v * F) * e * top \sqcap top * e^T * F * v^T \sqcap top * e^T * -F$
using *injective-comp-right-dist-inf assms(3) mult.semigroup-axioms semigroup.assoc* **by** *fastforce*
also have ... = $(-F \sqcap v * F) * e * top \sqcap top * e^T * (F * v^T \sqcap -F)$
using *injective-comp-right-dist-inf assms(3) conv-dist-comp inf.sup-monoid.add-assoc mult.semigroup-axioms univalent-comp-left-dist-inf semigroup.assoc* **by** *fastforce*
also have ... = $(-F \sqcap v * F) * e * top * top * e^T * (F * v^T \sqcap -F)$
by (*metis inf-top-right vector-export-comp vector-top-closed*)
also have ... = $(-F \sqcap v * F) * e * top * e^T * (F * v^T \sqcap -F)$
by (*simp add*: *comp-associative*)
also have ... $\leq (-F \sqcap v * F) * (F * v^T \sqcap -F)$
by (*smt assms(3) conv-dist-comp mult.semigroup-axioms mult-left-isotone shunt-bijective symmetric-top-closed top-right-mult-increasing semigroup.assoc*)
also have ... = $(-F \sqcap v * F) * ((v * F)^T \sqcap -F)$
by (*simp add*: *assms(1) conv-dist-comp*)
also have ... = $(-F \sqcap v * F) * (-F \sqcap v * F)^T$
using *assms(1) conv-complement conv-dist-inf* **by** (*simp add*: *inf.sup-monoid.add-commute*)
also have ... $\leq (-F \sqcap v) * (F \sqcap v)^\star * (F \sqcap v)^{T\star} * (-F \sqcap v)^T$
proof −
let *?Fv* = $F \sqcap v$
have $-F \sqcap v * F \leq -F \sqcap v * (F \sqcap v)^{T\star} * (F \sqcap v)^\star$
using *assms(5) inf.sup-right-isotone mult-right-isotone comp-associative* **by** *auto*
also have ... $\leq -F \sqcap v * (F \sqcap v)^\star$
proof −
have $v * v^T \leq 1$
by (*simp add*: *assms(2)*)
then have $v * v^T * F \leq F$
using *assms(1) dual-order.trans mult-left-isotone* **by** *blast*
then have $v * v^T * F^{T\star} * F^\star \leq F$
using *assms(1)* **by** (*metis mult-1-right preorder-idempotent star.circ-sup-one-right-unfold star.circ-transitive-equal star-one star-simulation-right-equal mult-assoc*)
then have $v * (F \sqcap v)^T * F^{T\star} * F^\star \leq F$
using *conv-isotone dual-order.trans inf.cobounded2 inf.sup-monoid.add-commute mult-left-isotone mult-right-isotone* **by** *presburger*
then have $v * (F \sqcap v)^T * (F \sqcap v)^{T\star} * (F \sqcap v)^\star \leq F$
using *conv-isotone dual-order.trans inf.cobounded2 inf.sup-monoid.add-commute mult-left-isotone mult-right-isotone* **by** (*meson comp-isotone conv-dist-inf inf.cobounded1 star-isotone*)
then have $-F \sqcap v * (F \sqcap v)^T * (F \sqcap v)^{T\star} * (F \sqcap v)^\star \leq bot$
using *eq-iff p-antitone pseudo-complement* **by** *auto*
then have $(-F \sqcap v * (F \sqcap v)^T * (F \sqcap v)^{T\star} * (F \sqcap v)^\star) \sqcup v * (v \sqcap F)^\star \leq v * (v \sqcap F)^\star$
using *bot-least le-bot* **by** *fastforce*
then have $(-F \sqcup v * (v \sqcap F)^\star) \sqcap (v * (F \sqcap v)^T * (F \sqcap v)^{T\star} * (F \sqcap v)^\star \sqcup v * (v \sqcap F)^\star) \leq v * (v \sqcap F)^\star$
by (*simp add*: *sup-inf-distrib2*)
then have $(-F \sqcup v * (v \sqcap F)^\star) \sqcap v * ((F \sqcap v)^T * (F \sqcap v)^{T\star} \sqcup 1) * (v \sqcap F)^\star \leq v * (v \sqcap F)^\star$
by (*simp add*: *inf.sup-monoid.add-commute mult.semigroup-axioms mult-left-dist-sup mult-right-dist-sup semigroup.assoc*)
then have $(-F \sqcup v * (v \sqcap F)^\star) \sqcap v * (F \sqcap v)^{T\star} * (v \sqcap F)^\star \leq v * (v \sqcap F)^\star$
by (*simp add*: *star-left-unfold-equal sup-commute*)
then have $-F \sqcap v * (F \sqcap v)^{T\star} * (v \sqcap F)^\star \leq v * (v \sqcap F)^\star$
using *comp-inf.mult-right-sub-dist-sup-left inf.order-lesseq-imp* **by** *blast*
thus *?thesis*
by (*simp add*: *inf.sup-monoid.add-commute*)
qed

```
    also have ... ≤ (v ⊓ −F ∗ (F ⊓ v)^T⋆) ∗ (F ⊓ v)⋆
      using dedekind-2 by (metis conv-star-commute inf.sup-monoid.add-commute)
    also have ... ≤ (v ⊓ −F ∗ F^T⋆) ∗ (F ⊓ v)⋆
      using conv-isotone inf.sup-right-isotone mult-left-isotone mult-right-isotone star-isotone by auto
    also have ... = (v ⊓ −F ∗ F) ∗ (F ⊓ v)⋆
      using assms(1) by (metis equivalence-comp-right-complement mult-left-one star-one
star-simulation-right-equal)
    also have ... = (−F ⊓ v) ∗ (F ⊓ v)⋆
      using assms(1) equivalence-comp-right-complement inf.sup-monoid.add-commute by auto
    finally have −F ⊓ v ∗ F ≤ (−F ⊓ v) ∗ (F ⊓ v)⋆
      by simp
    then have (−F ⊓ v ∗ F) ∗ (−F ⊓ v ∗ F)^T ≤ (−F ⊓ v) ∗ (F ⊓ v)⋆ ∗ ((−F ⊓ v) ∗ (F ⊓ v)⋆)^T
      by (simp add: comp-isotone conv-isotone)
    also have ... = (−F ⊓ v) ∗ (F ⊓ v)⋆ ∗ (F ⊓ v)^T⋆ ∗ (−F ⊓ v)^T
      by (simp add: comp-associative conv-dist-comp conv-star-commute)
    finally show ?thesis
      by simp
  qed
  also have ... ≤ (−F ⊓ v) ∗ ((F ⊓ v⋆) ⊔ (F ⊓ v^T⋆)) ∗ (−F ⊓ v)^T
  proof −
    have (F ⊓ v)⋆ ∗ (F ⊓ v)^T⋆ ≤ F⋆ ∗ F^T⋆
      using fc-isotone by auto
    also have ... ≤ F ∗ F
      by (metis assms(1) preorder-idempotent star.circ-sup-one-left-unfold star.circ-transitive-equal
star-right-induct-mult)
    finally have 21: (F ⊓ v)⋆ ∗ (F ⊓ v)^T⋆ ≤ F
      using assms(1) dual-order.trans by blast
    have (F ⊓ v)⋆ ∗ (F ⊓ v)^T⋆ ≤ v⋆ ∗ v^T⋆
      by (simp add: fc-isotone)
    then have (F ⊓ v)⋆ ∗ (F ⊓ v)^T⋆ ≤ F ⊓ v⋆ ∗ v^T⋆
      using 21 by simp
    also have ... = F ⊓ (v⋆ ⊔ v^T⋆)
      by (simp add: assms(2) cancel-separate-eq)
    finally show ?thesis
      by (metis assms(4) assms(6) comp-associative comp-inf.semiring.distrib-left comp-isotone
inf-pp-semi-commute mult-left-isotone regular-closed-inf)
  qed
  also have ... ≤ (−F ⊓ v) ∗ (F ⊓ v^T⋆) ∗ (−F ⊓ v)^T ⊔ (−F ⊓ v) ∗ (F ⊓ v⋆) ∗ (−F ⊓ v)^T
    by (simp add: mult-left-dist-sup mult-right-dist-sup)
  also have ... ≤ (−F ⊓ v) ∗ (−F ⊓ v)^T ⊔ (−F ⊓ v) ∗ (−F ⊓ v)^T
  proof −
    have (−F ⊓ v) ∗ (F ⊓ v^T⋆) ≤ (−F ⊓ v) ∗ ((F ⊓ v)^T⋆ ∗ (F ⊓ v)⋆ ⊓ v^T⋆)
      by (simp add: assms(5) inf.coboundedI1 mult-right-isotone)
    also have ... = (−F ⊓ v) ∗ ((F ⊓ v)^T ∗ (F ⊓ v)^T⋆ ∗ (F ⊓ v)⋆ ⊓ v^T⋆) ⊔ (−F ⊓ v) ∗ ((F ⊓ v)⋆ ⊓
v^T⋆)
      by (metis comp-associative comp-inf.mult-right-dist-sup mult-left-dist-sup star.circ-loop-fixpoint)
    also have ... ≤ (−F ⊓ v) ∗ (F ⊓ v)^T ∗ top ⊔ (−F ⊓ v) ∗ ((F ⊓ v)⋆ ⊓ v^T⋆)
      by (simp add: comp-associative comp-isotone inf.coboundedI2 inf.sup-monoid.add-commute
le-supI1)
    also have ... ≤ (−F ⊓ v) ∗ (F ⊓ v)^T ∗ top ⊔ (−F ⊓ v) ∗ (v⋆ ⊓ v^T⋆)
      by (smt comp-inf.mult-right-isotone comp-inf.semiring.add-mono eq-iff inf.cobounded2
inf.sup-monoid.add-commute mult-right-isotone star-isotone)
    also have ... ≤ bot ⊔ (−F ⊓ v) ∗ (v⋆ ⊓ v^T⋆)
      using assms(1, 2) forests-bot-1 by (metis comp-associative comp-inf.semiring.add-right-mono
mult-semi-associative vector-bot-closed)
    also have ... ≤ −F ⊓ v
      by (simp add: assms(2) acyclic-star-inf-conv)
    finally have 22: (−F ⊓ v) ∗ (F ⊓ v^T⋆) ≤ −F ⊓ v
```

```
      by simp
    have ((−F ⊓ v) ∗ (F ⊓ v^T⋆))^T = (F ⊓ v⋆) ∗ (−F ⊓ v)^T
      by (simp add: assms(1) conv-dist-inf conv-star-commute conv-dist-comp)
    then have (F ⊓ v⋆) ∗ (−F ⊓ v)^T ≤ (−F ⊓ v)^T
      using 22 conv-isotone by fastforce
    thus ?thesis
      using 22 by (metis assms(4) assms(6) comp-associative comp-inf.pp-comp-semi-commute
comp-inf.semiring.add-mono comp-isotone inf-pp-commute mult-left-isotone)
  qed
  also have ... = (−F ⊓ v) ∗ (−F ⊓ v)^T
    by simp
  also have ... ≤ v ∗ v^T
    by (simp add: comp-isotone conv-isotone)
  also have ... ≤ 1
    by (simp add: assms(2))
  thus ?thesis
    using calculation dual-order.trans by blast
qed
have 3: top ∗ ?i ∗ top = top
proof −
  have 31: regular (e^T ∗ −F ∗ v ∗ F ∗ e)
    using assms(3) assms(4) assms(6) arc-regular regular-mult-closed by auto
  have 32: bijective ((top ∗ e^T)^T)
    using assms(3) by (simp add: conv-dist-comp)
  have top ∗ ?i ∗ top = top ∗ (v ⊓ −F ∗ e ∗ top) ∗ ((top ∗ e^T ∗ F)^T ⊓ top)
    by (simp add: comp-associative comp-inf-vector-1)
  also have ... = (top ⊓ (−F ∗ e ∗ top)^T) ∗ v ∗ ((top ∗ e^T ∗ F)^T ⊓ top)
    using comp-inf-vector conv-dist-comp by auto
  also have ... = (−F ∗ e ∗ top)^T ∗ v ∗ (top ∗ e^T ∗ F)^T
    by simp
  also have ... = top^T ∗ e^T ∗ −F^T ∗ v ∗ F^T ∗ e^TT ∗ top^T
    by (simp add: comp-associative conv-complement conv-dist-comp)
  finally have 33: top ∗ ?i ∗ top = top ∗ e^T ∗ −F ∗ v ∗ F ∗ e ∗ top
    by (simp add: assms(1))
  have top ∗ ?i ∗ top ≠ bot
  proof (rule ccontr)
    assume ¬ top ∗ (v ⊓ − F ∗ e ∗ top ⊓ top ∗ e^T ∗ F) ∗ top ≠ bot
    then have top ∗ e^T ∗ −F ∗ v ∗ F ∗ e ∗ top = bot
      using 33 by auto
    then have e^T ∗ −F ∗ v ∗ F ∗ e = bot
      using 31 tarski comp-associative le-bot by fastforce
    then have top ∗ (−F ∗ v ∗ F ∗ e)^T ≤ −(e^T)
      by (metis comp-associative conv-complement-sub-leq conv-involutive p-bot schroeder-5-p)
    then have top ∗ e^T ∗ F^T ∗ v^T ∗ −F^T ≤ −(e^T)
      by (simp add: comp-associative conv-complement conv-dist-comp)
    then have v ∗ F ∗ e ∗ top ∗ e^T ≤ F
      by (metis assms(1) assms(4) comp-associative conv-dist-comp schroeder-3-p
symmetric-top-closed)
    then have v ∗ F ∗ e ∗ top ∗ top ∗ e^T ≤ F
      by (simp add: comp-associative)
    then have v ∗ F ∗ e ∗ top ≤ F ∗ (top ∗ e^T)^T
      using 32 shunt-bijective by (metis comp-associative conv-involutive)
    then have v ∗ F ∗ e ∗ top ≤ F ∗ e ∗ top
      using comp-associative conv-dist-comp by auto
    then have v⋆ ∗ F ∗ e ∗ top ≤ F ∗ e ∗ top
      using comp-associative star-left-induct-mult-iff by auto
    then have e^T ∗ F ∗ e ∗ top ≤ F ∗ e ∗ top
      by (meson assms(9) mult-left-isotone order-trans)
```

```
      then have e^T * F * e * top * (e * top)^T ≤ F
        using 32 shunt-bijective assms(3) mult-assoc by auto
      then have 34: e^T * F * e * top * top * e^T ≤ F
        by (metis conv-dist-comp mult.semigroup-axioms symmetric-top-closed semigroup.assoc)
      then have e^T ≤ F
      proof -
        have e^T ≤ e^T * e * e^T
          by (metis conv-involutive ex231c)
        also have ... ≤ e^T * F * e * e^T
          using assms(1) comp-associative mult-left-isotone mult-right-isotone by fastforce
        also have ... ≤ e^T * F * e * top * top * e^T
          by (simp add: mult-left-isotone top-right-mult-increasing vector-mult-closed)
        finally show ?thesis
          using 34 by simp
      qed
      then have 35: e ≤ F
        using assms(1) conv-order by fastforce
      have top * (F * e)^T ≤ - e
        using assms(8) comp-associative schroeder-4-p by auto
      then have top * e^T * F ≤ - e
        by (simp add: assms(1) comp-associative conv-dist-comp)
      then have (top * e^T)^T * e ≤ - F
        using schroeder-3-p by auto
      then have e * top * e ≤ - F
        by (simp add: conv-dist-comp)
      then have e ≤ - F
        by (simp add: assms(3) arc-top-arc)
      then have e ≤ F ⊓ - F
        using 35 inf.boundedI by blast
      then have e = bot
        using bot-unique by auto
      thus False
        using assms(10) by auto
    qed
    thus ?thesis
      by (metis assms(3) assms(4) assms(6) arc-regular regular-closed-inf regular-closed-top
regular-conv-closed regular-mult-closed semiring.mult-not-zero tarski)
  qed
  have bijective (?i * top) ∧ bijective (?i^T * top)
    using 1 2 3 arc-expanded by blast
  thus ?thesis
    by blast
qed

lemma exists-a-w:
  assumes symmetric g
    and forest f
    and f ≤ --g
    and regular f
    and (∃ w . minimum-spanning-forest w g ∧ f ≤ w ⊔ w^T)
    and vector j
    and regular j
    and forest h
    and forest-components h ≤ forest-components f
    and big-forest (forest-components h) d
    and d * top ≤ - j
    and forest-components h * j = j
    and forest-components f = (forest-components h * (d ⊔ d^T))^⋆ * forest-components h
```

and $f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$
and $(\forall\ a\ b\ .\ bf\text{-}between\text{-}arcs\ a\ b\ (forest\text{-}components\ h)\ d \wedge a \leq -(forest\text{-}components\ h) \sqcap -- g \wedge b \leq d \longrightarrow sum(b \sqcap g) \leq sum(a \sqcap g))$
and *regular d*
and *selected-edge h j g* $\leq -$ *forest-components f*
and *selected-edge h j g* $\neq$ *bot*
and $j \neq bot$
and *regular h*
and $h \leq --g$
shows $\exists w.$ *minimum-spanning-forest w g* $\wedge$
$f \sqcap - (selected\text{-}edge\ h\ j\ g)^T \sqcap - (path\ f\ h\ j\ g) \sqcup (f \sqcap - (selected\text{-}edge\ h\ j\ g)^T \sqcap (path\ f\ h\ j\ g))^T \sqcup (selected\text{-}edge\ h\ j\ g) \leq w \sqcup w^T$
proof −
let *?p* = *path f h j g*
let *?e* = *selected-edge h j g*
let $?f = (f \sqcap -?e^T \sqcap -?p) \sqcup (f \sqcap -?e^T \sqcap\ ?p)^T \sqcup\ ?e$
let *?F* = *forest-components f*
let *?H* = *forest-components h*
let *?ec* = *choose-component* (*forest-components h*) $j * -$ *choose-component* (*forest-components h*) $j^T \sqcap g$
from *assms*(*4*) **obtain** *w* **where** *2*: *minimum-spanning-forest w g* $\wedge$ $f \leq w \sqcup w^T$
using *assms*(*5*) **by** *blast*
hence *3*: *regular w* $\wedge$ *regular f* $\wedge$ *regular ?e*
using *assms*(*1*, *4*) *boruvka-inner-invariant-def boruvka-outer-invariant-def minarc-regular minimum-spanning-forest-def spanning-forest-def* **by** *metis*
have *5*: *equivalence ?F*
using *assms*(*2*) *forest-components-equivalence* **by** *auto*
have $?e^T * top * ?e^T = ?e^T$
using *assms*(*4*) **by** (*metis arc-conv-closed arc-top-arc coreflexive-bot-closed coreflexive-symmetric minarc-arc minarc-bot-iff semiring.mult-not-zero*)
hence $?e^T * top * ?e^T \leq -?F$
using *5 assms*(*17*) *conv-complement conv-isotone* **by** *fastforce*
hence *6*: $?e * ?F * ?e = bot$
using *assms*(*2*) *le-bot triple-schroeder-p* **by** *simp*
let $?q = w \sqcap top * ?e * w^{T\star}$
let $?v = (w \sqcap -(top * ?e * w^{T\star})) \sqcup ?q^T$
have *7*: *regular ?q*
using *3 regular-closed-star regular-conv-closed regular-mult-closed* **by** *auto*
have *8*: *injective ?v*
proof (*rule kruskal-exchange-injective-inv-1*)
show *injective w*
using *2 minimum-spanning-forest-def spanning-forest-def* **by** *blast*
next
show *covector* $(top * ?e * w^{T\star})$
by (*simp add*: *covector-mult-closed*)
next
show $top * ?e * w^{T\star} * w^T \leq top * ?e * w^{T\star}$
by (*simp add*: *mult-right-isotone star.right-plus-below-circ mult-assoc*)
next
show *coreflexive* $((top * ?e * w^{T\star})^T * (top * ?e * w^{T\star}) \sqcap w^T * w)$
by (*metis 2 comp-inf.semiring.mult-not-zero forest-bot kruskal-injective-inv-3 minarc-arc minarc-bot-iff minimum-spanning-forest-def semiring.mult-not-zero spanning-forest-def*)
qed
have *9*: *components g* $\leq$ *forest-components ?v*
proof (*rule kruskal-exchange-spanning-inv-1*)
show *injective* $(w \sqcap - (top * ?e * w^{T\star}) \sqcup (w \sqcap top * ?e * w^{T\star})^T)$
using *8* **by** *simp*
next

```
    show regular (w ⊓ top * ?e * w^T⋆)
      using 7 by simp
  next
    show components g ≤ forest-components w
      using 2 minimum-spanning-forest-def spanning-forest-def by blast
  qed
  have 10: spanning-forest ?v g
  proof (unfold spanning-forest-def, intro conjI)
    show injective ?v
      using 8 by auto
  next
    show acyclic ?v
    proof (rule kruskal-exchange-acyclic-inv-1)
      show pd-kleene-allegory-class.acyclic w
        using 2 minimum-spanning-forest-def spanning-forest-def by blast
    next
      show covector (top * ?e * w^T⋆)
        by (simp add: covector-mult-closed)
    qed
  next
    show ?v ≤ −−g
    proof (rule sup-least)
      show w ⊓ − (top * ?e * w^T⋆) ≤ − − g
        using 7 inf.coboundedI1 minimum-spanning-forest-def spanning-forest-def 2 by blast
    next
      show (w ⊓ top * ?e * w^T⋆)^T ≤ − − g
        by (metis assms(1) 2 conv-complement conv-isotone inf.coboundedI1
minimum-spanning-forest-def spanning-forest-def)
    qed
  next
    show components g ≤ forest-components ?v
      using 9 by simp
  next
    show regular ?v
      using 3 regular-closed-star regular-conv-closed regular-mult-closed by auto
  qed
  have 11: sum (?v ⊓ g) = sum (w ⊓ g)
  proof −
    have sum (?v ⊓ g) = sum (w ⊓ −(top * ?e * w^T⋆) ⊓ g) + sum (?q^T ⊓ g)
      using 2 conv-complement conv-top epm-8 inf-import-p inf-top-right regular-closed-top
vector-top-closed minimum-spanning-forest-def spanning-forest-def sum-disjoint by smt
    also have ... = sum (w ⊓ −(top * ?e * w^T⋆) ⊓ g) + sum (?q ⊓ g)
      by (simp add: assms(1) sum-symmetric)
    also have ... = sum (((w ⊓ −(top * ?e * w^T⋆)) ⊔ ?q) ⊓ g)
      using inf-commute inf-left-commute sum-disjoint by simp
    also have ... = sum (w ⊓ g)
      using 3 7 8 maddux-3-11-pp by auto
    finally show ?thesis
      by simp
  qed
  have 12: ?v ⊔ ?v^T = w ⊔ w^T
  proof −
    have ?v ⊔ ?v^T = (w ⊓ −?q) ⊔ ?q^T ⊔ (w^T ⊓ −?q^T) ⊔ ?q
      using conv-complement conv-dist-inf conv-dist-sup inf-import-p sup-assoc by simp
    also have ... = w ⊔ w^T
      using 3 7 conv-complement conv-dist-inf inf-import-p maddux-3-11-pp sup-monoid.add-assoc
sup-monoid.add-commute by auto
    finally show ?thesis
```

```
    by simp
qed
have 13: ?v * ?e^T = bot
proof (rule kruskal-reroot-edge)
  show injective (?e^T * top)
    using assms(18) minarc-arc minarc-bot-iff by blast
next
  show pd-kleene-allegory-class.acyclic w
    using 2 minimum-spanning-forest-def spanning-forest-def by simp
qed
have ?v ⊓ ?e ≤ ?v ⊓ top * ?e
  using inf.sup-right-isotone top-left-mult-increasing by simp
also have ... ≤ ?v * (top * ?e)^T
  using covector-restrict-comp-conv covector-mult-closed vector-top-closed by simp
finally have 14: ?v ⊓ ?e = bot
  using 13 by (metis conv-dist-comp mult-assoc le-bot mult-left-zero)
let ?i = ?v ⊓ (- ?F) * ?e * top ⊓ top * ?e^T * ?F
let ?w = (?v ⊓ -?i) ⊔ ?e
have 15: regular ?i
  using 3 regular-closed-star regular-conv-closed regular-mult-closed by simp
have 16: ?F ≤ -?i
proof -
  have 161: bijective (?e * top)
    using assms(18) minarc-arc minarc-bot-iff by auto
  have ?i ≤ - ?F * ?e * top
    using inf.cobounded2 inf.coboundedI1 by blast
  also have ... = - (?F * ?e * top)
    using 161 comp-bijective-complement by (simp add: mult-assoc)
  finally have ?i ≤ - (?F * ?e * top)
    by blast
  then have 162: ?i ⊓ ?F ≤ - (?F * ?e * top)
    using inf.coboundedI1 by blast
  have ?i ⊓ ?F ≤ ?F ⊓ (top * ?e^T * ?F)
    by (meson inf-le1 inf-le2 le-infI order-trans)
  also have ... ≤ ?F * (top * ?e^T * ?F)^T
    by (simp add: covector-mult-closed covector-restrict-comp-conv)
  also have ... = ?F * ?F^T * ?e^TT * top^T
    by (simp add: conv-dist-comp mult-assoc)
  also have ... = ?F * ?F * ?e * top
    by (simp add: conv-dist-comp conv-star-commute)
  also have ... = ?F * ?e * top
    by (simp add: 5 preorder-idempotent)
  finally have ?i ⊓ ?F ≤ ?F * ?e * top
    by simp
  then have ?i ⊓ ?F ≤ ?F * ?e * top ⊓ - (?F * ?e * top)
    using 162 inf.bounded-iff by blast
  also have ... = bot
    by simp
  finally show ?thesis
    using le-bot p-antitone-iff pseudo-complement by blast
qed
have 17: ?i ≤ top * ?e^T * (?F ⊓ ?v ⊓ -?i)^T⋆
proof -
  have ?i ≤ ?v ⊓ - ?F * ?e * top ⊓ top * ?e^T * (?F ⊓ ?v)^T⋆ * (?F ⊓ ?v)⋆
    by (smt 2 8 12 inf.sup-right-isotone kruskal-forest-components-inf mult-right-isotone mult-assoc)
  also have ... = ?v ⊓ - ?F * ?e * top ⊓ top * ?e^T * (?F ⊓ ?v)^T⋆ * (1 ⊔ (?F ⊓ ?v)⋆ * (?F ⊓ ?v))
    using star-left-unfold-equal star.circ-right-unfold-1 by auto
  also have ... = ?v ⊓ - ?F * ?e * top ⊓ (top * ?e^T * (?F ⊓ ?v)^T⋆ ⊔ top * ?e^T * (?F ⊓ ?v)^T⋆ *
```

$(?F \sqcap ?v)^\star * (?F \sqcap ?v))$
by (*simp add*: *mult-left-dist-sup mult-assoc*)
also have ... = $(?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * (?F \sqcap ?v)^{T\star}) \sqcup (?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * (?F \sqcap ?v)^{T\star} * (?F \sqcap ?v)^\star * (?F \sqcap ?v))$
using *comp-inf.semiring.distrib-left* **by** *blast*
also have ... $\leq top * ?e^T * (?F \sqcap ?v)^{T\star} \sqcup (?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * (?F \sqcap ?v)^{T\star} * (?F \sqcap ?v)^\star * (?F \sqcap ?v))$
using *comp-inf.semiring.add-right-mono inf-le2* **by** *blast*
also have ... $\leq top * ?e^T * (?F \sqcap ?v)^{T\star} \sqcup (?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * (?F^T \sqcap ?v^T)^\star * (?F \sqcap ?v)^\star * (?F \sqcap ?v))$
by (*simp add*: *conv-dist-inf*)
also have ... $\leq top * ?e^T * (?F \sqcap ?v)^{T\star} \sqcup (?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * ?F^{T\star} * ?F^\star * (?F \sqcap ?v))$
proof −
have $top * ?e^T * (?F^T \sqcap ?v^T)^\star * (?F \sqcap ?v)^\star * (?F \sqcap ?v) \leq top * ?e^T * ?F^{T\star} * ?F^\star * (?F \sqcap ?v)$
using *star-isotone* **by** (*simp add*: *comp-isotone*)
then have $?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * (?F^T \sqcap ?v^T)^\star * (?F \sqcap ?v)^\star * (?F \sqcap ?v) \leq ?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * ?F^{T\star} * ?F^\star * (?F \sqcap ?v)$
using *inf.sup-right-isotone* **by** *blast*
thus *?thesis*
using *sup-right-isotone* **by** *blast*
qed
also have ... = $top * ?e^T * (?F \sqcap ?v)^{T\star} \sqcup (?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * ?F^\star * ?F^\star * (?F \sqcap ?v))$
using *5* **by** *auto*
also have ... = $top * ?e^T * (?F \sqcap ?v)^{T\star} \sqcup (?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * ?F * ?F * (?F \sqcap ?v))$
by (*simp add*: *assms*(*2*) *forest-components-star*)
also have ... = $top * ?e^T * (?F \sqcap ?v)^{T\star} \sqcup (?v \sqcap - ?F * ?e * top \sqcap top * ?e^T * ?F * (?F \sqcap ?v))$
using *5 mult.semigroup-axioms preorder-idempotent semigroup.assoc* **by** *fastforce*
also have ... = $top * ?e^T * (?F \sqcap ?v)^{T\star}$
proof −
have $?e * top * ?e^T \leq 1$
using *assms*(*18*) *arc-expanded minarc-arc minarc-bot-iff* **by** *auto*
then have $?F * ?e * top * ?e^T \leq ?F * 1$
by (*metis comp-associative comp-isotone mult-semi-associative star.circ-transitive-equal*)
then have $?v * ?v^T * ?F * ?e * top * ?e^T \leq 1 * ?F * 1$
using *8* **by** (*smt comp-isotone mult-assoc*)
then have *171*: $?v * ?v^T * ?F * ?e * top * ?e^T \leq ?F$
by *simp*
then have $?v * (?F \sqcap ?v)^T * ?F * ?e * top * ?e^T \leq ?F$
proof −
have $?v * (?F \sqcap ?v)^T * ?F * ?e * top * ?e^T \leq ?v * ?v^T * ?F * ?e * top * ?e^T$
by (*simp add*: *conv-dist-inf mult-left-isotone mult-right-isotone*)
thus *?thesis*
using *171 order-trans* **by** *blast*
qed
then have *172*: $-?F * ((?F \sqcap ?v)^T * ?F * ?e * top * ?e^T)^T \leq -?v$
using *schroeder-4-p* **by** (*smt comp-associative order-lesseq-imp pp-increasing*)
have $-?F * ((?F \sqcap ?v)^T * ?F * ?e * top * ?e^T)^T = -?F * ?e^{TT} * top^T * ?e^T * ?F^T * (?F \sqcap ?v)^{TT}$
by (*simp add*: *comp-associative conv-dist-comp*)
also have ... = $-?F * ?e * top * ?e^T * ?F * (?F \sqcap ?v)$
using *5* **by** *auto*
also have ... = $-?F * ?e * top * top * ?e^T * ?F * (?F \sqcap ?v)$
using *comp-associative* **by** *auto*
also have ... = $-?F * ?e * top \sqcap top * ?e^T * ?F * (?F \sqcap ?v)$
using *comp-associative comp-inf.star.circ-decompose-9 comp-inf.star-star-absorb*

comp-inf-covector inf-vector-comp vector-top-closed **by** *smt*
finally have $-?F * ((?F \sqcap ?v)^T * ?F * ?e * top * ?e^T)^T = -?F * ?e * top \sqcap top * ?e^T * ?F * (?F \sqcap ?v)$
by *simp*
then have $-?F * ?e * top \sqcap top * ?e^T * ?F * (?F \sqcap ?v) \leq -?v$
using *172* **by** *auto*
then have $?v \sqcap -?F * ?e * top \sqcap top * ?e^T * ?F * (?F \sqcap ?v) \leq bot$
by (*smt bot-unique inf.sup-monoid.add-commute p-shunting-swap pseudo-complement*)
thus *?thesis*
using *le-bot sup-monoid.add-0-right* **by** *blast*
qed
also have $... = top * ?e^T * (?F \sqcap ?v \sqcap -?i)^{T\star}$
by (*smt 16 comp-inf.coreflexive-comp-inf-complement inf-top-right p-bot pseudo-complement top.extremum*)
finally show *?thesis*
by *blast*
qed
have *18*: $?i \leq top * ?e^T * ?w^{T\star}$
proof −
have $?i \leq top * ?e^T * (?F \sqcap ?v \sqcap -?i)^{T\star}$
using *17* **by** *simp*
also have $... \leq top * ?e^T * (?v \sqcap -?i)^{T\star}$
using *mult-right-isotone conv-isotone star-isotone inf.cobounded2 inf.sup-monoid.add-assoc* **by** *presburger*
also have $... \leq top * ?e^T * ((?v \sqcap -?i) \sqcup ?e)^{T\star}$
using *mult-right-isotone conv-isotone star-isotone sup-ge1* **by** *simp*
finally show *?thesis*
by *blast*
qed
have *19*: $?i \leq top * ?e^T * ?v^{T\star}$
proof −
have $?i \leq top * ?e^T * (?F \sqcap ?v \sqcap -?i)^{T\star}$
using *17* **by** *simp*
also have $... \leq top * ?e^T * (?v \sqcap -?i)^{T\star}$
using *mult-right-isotone conv-isotone star-isotone inf.cobounded2 inf.sup-monoid.add-assoc* **by** *presburger*
also have $... \leq top * ?e^T * (?v)^{T\star}$
using *mult-right-isotone conv-isotone star-isotone* **by** *auto*
finally show *?thesis*
by *blast*
qed
have *20*: $f \sqcup f^T \leq (?v \sqcap -?i \sqcap -?i^T) \sqcup (?v^T \sqcap -?i \sqcap -?i^T)$
proof (*rule kruskal-edge-between-components-2*)
show $?F \leq - ?i$
using *16* **by** *simp*
next
show *injective f*
by (*simp add*: *assms*(*2*))
next
show $f \sqcup f^T \leq w \sqcap - (top * ?e * w^{T\star}) \sqcup (w \sqcap top * ?e * w^{T\star})^T \sqcup (w \sqcap - (top * ?e * w^{T\star}) \sqcup (w \sqcap top * ?e * w^{T\star})^T)^T$
using *2 12* **by** (*metis conv-dist-sup conv-involutive conv-isotone le-supI sup-commute*)
qed
have *minimum-spanning-forest ?w g* $\wedge$ $?f \leq ?w \sqcup ?w^T$
proof (*intro conjI*)
have *211*: $?e^T \leq ?v^\star$
proof (*rule kruskal-edge-arc-1*[**where** *g*=*g* **and** *h*=*?ec*])
show $?e \leq -- ?ec$

```
    using minarc-below by blast
  next
    show ?ec ≤ g
    using assms(4) inf.cobounded2 by (simp add: boruvka-inner-invariant-def
boruvka-outer-invariant-def conv-dist-inf)
  next
    show symmetric g
    by (meson assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def)
  next
    show components g ≤ forest-components (w ⊓ − (top * ?e * w^T⋆) ⊔ (w ⊓ top * ?e * w^T⋆)^T)
    using 9 by simp
  next
    show (w ⊓ − (top * ?e * w^T⋆) ⊔ (w ⊓ top * ?e * w^T⋆)^T) * ?e^T = bot
    using 13 by blast
  qed
  have 212: arc ?i
  proof (rule boruvka-edge-arc)
    show equivalence ?F
    by (simp add: 5)
  next
    show forest ?v
    using 10 spanning-forest-def by blast
  next
    show arc ?e
    using assms(18) minarc-arc minarc-bot-iff by blast
  next
    show regular ?F
    using 3 regular-closed-star regular-conv-closed regular-mult-closed by auto
  next
    show ?F ≤ forest-components (?F ⊓ ?v)
    by (simp add: 12 2 8 kruskal-forest-components-inf)
  next
    show regular ?v
    using 10 spanning-forest-def by blast
  next
    show ?v * ?e^T = bot
    using 13 by auto
  next
    show ?e * ?F * ?e = bot
    by (simp add: 6)
  next
    show ?e^T ≤ ?v⋆
    using 211 by auto

  next
    show ?e ≠ bot
    by (simp add: assms(18))
  qed
  show minimum-spanning-forest ?w g
  proof (unfold minimum-spanning-forest-def, intro conjI)
    have (?v ⊓ −?i) * ?e^T ≤ ?v * ?e^T
    using inf-le1 mult-left-isotone by simp
    hence (?v ⊓ −?i) * ?e^T = bot
    using 13 le-bot by simp
    hence 221: ?e * (?v ⊓ −?i)^T = bot
    using conv-dist-comp conv-involutive conv-bot by force
    have 222: injective ?w
    proof (rule injective-sup)
```

```
    show injective (?v ⊓ − ?i)
      using 8 by (simp add: injective-inf-closed)
  next
    show coreflexive (?e ∗ (?v ⊓ − ?i)^T)
      using 221 by simp
  next
    show injective ?e
      using assms(4) arc-injective minarc-arc by (metis coreflexive-bot-closed coreflexive-injective
minarc-bot-iff)
  qed
  show spanning-forest ?w g
  proof (unfold spanning-forest-def, intro conjI)
    show injective ?w
      using 222 by simp
  next
    show acyclic ?w
    proof (rule kruskal-exchange-acyclic-inv-2)
      show acyclic ?v
        using 10 spanning-forest-def by blast
    next
      show injective ?v
        using 8 by simp
    next
      show ?i ≤ ?v
        using inf.coboundedI1 by simp
    next
      show bijective (?i^T ∗ top)
        using 212 by simp
    next
      show bijective (?e ∗ top)
        using assms(4) by (smt 14 212 comp-inf.idempotent-bot-closed conv-complement minarc-arc
minarc-bot-iff p-bot regular-closed-bot semiring.mult-not-zero symmetric-top-closed)
    next
      show ?i ≤ top ∗ ?e^T ∗ ?v^T⋆
        using 19 by simp
    next
      show ?v ∗ ?e^T ∗ top = bot
        using 13 by simp
    qed
  next
    have ?w ≤ ?v ⊔ ?e
      using inf-le1 sup-left-isotone by simp
    also have ... ≤ −−g ⊔ ?e
      using 10 sup-left-isotone spanning-forest-def by blast
    also have ... ≤ −−g ⊔ −−h
    proof −
      have 1: −−g ≤ −−g ⊔ −−h
        by simp
      have 2: ?e ≤ −−g ⊔ −−h
        by (metis inf.coboundedI1 inf.sup-monoid.add-commute minarc-below order.trans p-dist-inf
p-dist-sup sup.cobounded1)
      then show ?thesis
        using 1 2 by simp
    qed
    also have ... ≤ −−g
      using assms(20, 21) by auto
    finally show ?w ≤ −−g
      by simp
```

```
    next
      have 223: ?i ≤ (?v ⊓ − ?i)^T⋆ ∗ ?e^T ∗ top
      proof (rule boruvka-exchange-spanning-inv)
        show forest ?v
          using 10 spanning-forest-def by blast
      next
        show ?v⋆ ∗ ?e^T = ?e^T
          using 13 by (smt conv-complement conv-dist-comp conv-involutive conv-star-commute
dense-pp fc-top regular-closed-top star-absorb)
      next
        show ?i ≤ ?v ⊓ top ∗ ?e^T ∗ ?w^T⋆
          using 18 inf.sup-monoid.add-assoc by auto
      next
        show arc ?i
          using 212 by blast
      next
        show arc ?e
          using assms(18) minarc-arc minarc-bot-iff by auto
      next
        show ?v ≤ −−g
          using 10 spanning-forest-def by blast
      next
        show ?w ≤ −−g
        proof −
          have 2231: ?e ≤ −−g
            by (metis inf.boundedE minarc-below pp-dist-inf)
          have ?w ≤ ?v ⊔ ?e
            using inf-le1 sup-left-isotone by simp
          also have ... ≤ −−g
            using 2231 10 spanning-forest-def sup-least by blast
          finally show ?thesis
            by blast
        qed
      next
        show ?e ≤ −−g
          by (metis inf.boundedE minarc-below pp-dist-inf)
      next
        show components g ≤ forest-components ?v
          by (simp add: 9)
      qed
      have components g ≤ forest-components ?v
        using 10 spanning-forest-def by auto
      also have ... ≤ forest-components ?w
      proof (rule kruskal-exchange-forest-components-inv)
      next
        show injective ((?v ⊓ − ?i) ⊔ ?e)
          using 222 by simp
      next
        show regular ?i
          using 15 by simp
      next
        show ?e ∗ top ∗ ?e = ?e
          using assms(4) by (metis arc-top-arc minarc-arc minarc-bot-iff semiring.mult-not-zero)
      next
        show ?i ≤ top ∗ ?e^T ∗ ?v^T⋆
          using 19 by blast
      next
        show ?v ∗ ?e^T ∗ top = bot
```

```
        using 13 by simp
      next
        show injective ?v
          using 8 by simp
      next
        show ?i ≤ ?v
          by (simp add: le-infI1)
      next
        show ?i ≤ (?v ⊓ − ?i)^T⋆ ∗ ?e^T ∗ top
          using 223 by blast
      qed
      finally show components g ≤ forest-components ?w
        by simp
    next
      show regular ?w
        using 3 7 regular-conv-closed by simp
    qed
  next
    have 224: ?e ⊓ g ≠ bot
      using assms(18) inf.left-commute inf-bot-right minarc-meet-bot by fastforce
    have 225: sum (?e ⊓ g) ≤ sum (?i ⊓ g)
    proof (rule a-to-e-in-bigforest)
      show symmetric g
        using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
    next
      show j ≠ bot
        by (simp add: assms(19))
    next
      show f ≤ −− g
        by (simp add: assms(3))
    next
      show vector j
        using assms(6) boruvka-inner-invariant-def by blast
    next
      show forest h
        by (simp add: assms(8))
    next
      show  big-forest (forest-components h) d
        by (simp add: assms(10))
    next
      show f ⊔ f^T = h ⊔ h^T ⊔ d ⊔ d^T
        by (simp add: assms(14))
    next
      show ∀ a b. bf-between-arcs a b (?H) d ∧ a ≤ − ?H ⊓ − − g ∧ b ≤ d ⟶ sum (b ⊓ g) ≤ sum
(a ⊓ g)
        by (simp add: assms(15))
    next
      show regular d
        using assms(16) by auto
    next
      show ?e = ?e
        by simp
    next
      show arc ?i
        using 212 by blast
    next
      show bf-between-arcs ?i ?e ?H (d ⊔ ?e)
      proof −
```

have $d^T * ?H * ?e = bot$
using *assms(19) assms(11) assms(12) assms(6) assms(7) dT-He-eq-bot le-bot* **by** *blast*
then have *251*: $d^T * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
by *simp*
then have $d^T * ?H * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
by (*metis assms(8) forest-components-star star.circ-decompose-9 mult-assoc*)
then have $d^T * (?H * d)^\star * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
proof −
have $d^T * ?H * d \leq 1$
using *assms(10) big-forest-def dTransHd-le-1* **by** *blast*
then have $d^T * ?H * d * (?H * d)^\star * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
by (*metis mult-left-isotone star.circ-circ-mult star-involutive star-one*)
then have $d^T * ?H * ?e \sqcup d^T * ?H * d * (?H * d)^\star * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
using *251* **by** *simp*
then have $d^T * (1 \sqcup ?H * d * (?H * d)^\star) * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
by (*simp add*: *comp-associative comp-left-dist-sup semiring.distrib-right*)
thus *?thesis*
by (*simp add*: *star-left-unfold-equal*)
qed
then have $?H * d^T * (?H * d)^\star * ?H * ?e \leq ?H * (?H * d)^\star * ?H * ?e$
by (*simp add*: *mult-right-isotone mult-assoc*)
then have $?H * d^T * (?H * d)^\star * ?H * ?e \leq ?H * ?H * (d * ?H)^\star * ?e$
by (*smt star-slide mult-assoc*)
then have $?H * d^T * (?H * d)^\star * ?H * ?e \leq ?H * (d * ?H)^\star * ?e$
by (*metis assms(8) forest-components-star star.circ-decompose-9*)
then have $?H * d^T * (?H * d)^\star * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
using *star-slide* **by** *auto*
then have $?H * d * (?H * d)^\star * ?H * ?e \sqcup ?H * d^T * (?H * d)^\star * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
by (*smt le-supI star.circ-loop-fixpoint sup.cobounded2 sup-commute mult-assoc*)
then have $(?H * (d \sqcup d^T)) * (?H * d)^\star * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
by (*simp add*: *semiring.distrib-left semiring.distrib-right*)
then have $(?H * (d \sqcup d^T))^\star * (?H * d)^\star * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
by (*simp add*: *star-left-induct-mult mult-assoc*)
then have *252*: $(?H * (d \sqcup d^T))^\star * ?H * ?e \leq (?H * d)^\star * ?H * ?e$
by (*smt mult-left-dist-sup star.circ-transitive-equal star-slide star-sup-1 mult-assoc*)
have $?i \leq top * ?e^T * ?F$
by *auto*
then have $?i^T \leq ?F^T * ?e^{TT} * top^T$
by (*simp add*: *conv-dist-comp conv-dist-inf mult-assoc*)
then have $?i^T * top \leq ?F^T * ?e^{TT} * top^T * top$
using *comp-isotone* **by** *blast*
also have $... = ?F^T * ?e^{TT} * top^T$
by (*simp add*: *vector-mult-closed*)
also have $... = ?F * ?e^{TT} * top^T$
by (*simp add*: *conv-dist-comp conv-star-commute*)
also have $... = ?F * ?e * top$
by *simp*
also have $... = (?H * (d \sqcup d^T))^\star * ?H * ?e * top$
by (*simp add*: *assms(13)*)
also have $... \leq (?H * d)^\star * ?H * ?e * top$
by (*simp add*: *252 comp-isotone*)
also have $... \leq (?H * (d \sqcup ?e))^\star * ?H * ?e * top$
by (*simp add*: *comp-isotone star-isotone*)
finally have $?i^T * top \leq (?H * (d \sqcup ?e))^\star * ?H * ?e * top$
by *blast*
thus *?thesis*
using *212 assms(18) bf-between-arcs-def minarc-arc minarc-bot-iff* **by** *blast*

```
      qed
    next
      show ?i ≤ - ?H ⊓ -- g
      proof -
        have 241: ?i ≤ -?H
          using 16 assms(9) inf.order-lesseq-imp p-antitone-iff by blast
        have ?i ≤ -- g
          using 10 inf.coboundedI1 spanning-forest-def by blast
        thus ?thesis
          using 241 inf-greatest by blast
      qed
    next
      show regular h
        using assms(20) by auto
    qed
    have ?v ⊓ ?e ⊓ -?i = bot
      using 14 by simp
    hence sum (?w ⊓ g) = sum (?v ⊓ -?i ⊓ g) + sum (?e ⊓ g)
      using sum-disjoint inf-commute inf-assoc by simp
    also have ... ≤ sum (?v ⊓ -?i ⊓ g) + sum (?i ⊓ g)
      using 224 225 sum-plus-right-isotone by simp
    also have ... = sum (((?v ⊓ -?i) ⊔ ?i) ⊓ g)
      using sum-disjoint inf-le2 pseudo-complement by simp
    also have ... = sum ((?v ⊔ ?i) ⊓ (-?i ⊔ ?i) ⊓ g)
      by (simp add: sup-inf-distrib2)
    also have ... = sum ((?v ⊔ ?i) ⊓ g)
      using 15 by (metis inf-top-right stone)
    also have ... = sum (?v ⊓ g)
      by (simp add: inf.sup-monoid.add-assoc)
    finally have sum (?w ⊓ g) ≤ sum (?v ⊓ g)
      by simp
    thus ∀ u . spanning-forest u g ⟶ sum (?w ⊓ g) ≤ sum (u ⊓ g)
      using 2 11 minimum-spanning-forest-def by auto
  qed
next
  have ?f ≤ f ⊔ f^T ⊔ ?e
    using conv-dist-inf inf-le1 sup-left-isotone sup-mono by (smt inf.order-lesseq-imp)
  also have ... ≤ (?v ⊓ -?i ⊓ -?i^T) ⊔ (?v^T ⊓ -?i ⊓ -?i^T) ⊔ ?e
    using 20 sup-left-isotone by simp
  also have ... ≤ (?v ⊓ -?i) ⊔ (?v^T ⊓ -?i ⊓ -?i^T) ⊔ ?e
    using inf.cobounded1 sup-inf-distrib2 by presburger
  also have ... = ?w ⊔ (?v^T ⊓ -?i ⊓ -?i^T)
    by (simp add: sup-assoc sup-commute)
  also have ... ≤ ?w ⊔ (?v^T ⊓ -?i^T)
    using inf.sup-right-isotone inf-assoc sup-right-isotone by simp
  also have ... ≤ ?w ⊔ ?w^T
    using conv-complement conv-dist-inf conv-dist-sup sup-right-isotone by simp
  finally show ?f ≤ ?w ⊔ ?w^T
    by simp
qed
thus ?thesis by auto
qed

lemma boruvka-outer-invariant-when-e-not-bot:
  assumes boruvka-inner-invariant j f h g d
    and j ≠ bot
    and selected-edge h j g ≤ - forest-components f
    and selected-edge h j g ≠ bot
```

shows *boruvka-outer-invariant* (*f* ⊓ − *selected-edge h j g*^{T} ⊓ − *path f h j g* ⊔ (*f* ⊓ − *selected-edge h j g*^{T} ⊓ *path f h j g*)^T ⊔ *selected-edge h j g*) *g*
proof −
let *?c* = *choose-component* (*forest-components h*) *j*
let *?p* = *path f h j g*
let *?F* = *forest-components f*
let *?H* = *forest-components h*
let *?e* = *selected-edge h j g*
let *?f'* = *f* ⊓ −*?e*^{T} ⊓ −*?p* ⊔ (*f* ⊓ −*?e*^{T} ⊓ *?p*)^T ⊔ *?e*
let *?d'* = *d* ⊔ *?e*
let *?j'* = *j* ⊓ −*?c*
show *boruvka-outer-invariant ?f' g*
proof (*unfold boruvka-outer-invariant-def*, *intro conjI*)
show *symmetric g*
by (*meson assms*(*1*) *boruvka-inner-invariant-def boruvka-outer-invariant-def*)
next
show *injective ?f'*
proof (*rule kruskal-injective-inv*)
show *injective* (*f* ⊓ − *?e*^{T})
by (*meson assms*(*1*) *boruvka-inner-invariant-def boruvka-outer-invariant-def injective-inf-closed*)
show *covector* (*?p*)
using *covector-mult-closed* **by** *simp*
show *?p* ∗ (*f* ⊓ − *?e*^{T})^T ≤ *?p*
by (*simp add*: *mult-right-isotone star.left-plus-below-circ star-plus mult-assoc*)
show *?e* ≤ *?p*
by (*meson mult-left-isotone order.trans star-outer-increasing top.extremum*)
show *?p* ∗ (*f* ⊓ − *?e*^{T})^T ≤ − *?e*
proof −
have *?p* ∗ (*f* ⊓ − *?e*^{T})^T ≤ *?p* ∗ *f*^{T}
by (*simp add*: *conv-dist-inf mult-right-isotone*)
also have ... ≤ *top* ∗ *?e* ∗ (*f*)^{T*} ∗ *f*^{T}
using *conv-dist-inf star-isotone comp-isotone* **by** *simp*
also have ... ≤ − *?e*
using *assms*(*1*) *boruvka-inner-invariant-def assms*(*4*) *boruvka-outer-invariant-def kruskal-injective-inv-2 minarc-arc minarc-bot-iff* **by** *auto*
finally show *?thesis* **.**
qed
show *injective* (*?e*)
by (*metis arc-injective coreflexive-bot-closed minarc-arc minarc-bot-iff semiring.mult-not-zero*)
show *coreflexive* (*?p*^{T} ∗ *?p* ⊓ (*f* ⊓ − *?e*^{T})^T ∗ (*f* ⊓ − *?e*^{T}))
proof −
have (*?p*^{T} ∗ *?p* ⊓ (*f* ⊓ − *?e*^{T})^T ∗ (*f* ⊓ − *?e*^{T})) ≤ *?p*^{T} ∗ *?p* ⊓ *f*^{T} ∗ *f*
using *conv-dist-inf inf.sup-right-isotone mult-isotone* **by** *simp*
also have ... ≤ (*top* ∗ *?e* ∗ *f*^{T*})^T ∗ (*top* ∗ *?e* ∗ *f*^{T*}) ⊓ *f*^{T} ∗ *f*
by (*metis comp-associative comp-inf.coreflexive-transitive comp-inf.mult-right-isotone comp-isotone conv-isotone inf.cobounded1 inf.idem inf.sup-monoid.add-commute star-isotone top.extremum*)
also have ... ≤ *1*
using *assms*(*1*) *assms*(*4*) *boruvka-inner-invariant-def boruvka-outer-invariant-def kruskal-injective-inv-3 minarc-arc minarc-bot-iff* **by** *auto*
finally show *?thesis*
by *simp*
qed
qed
next
show *acyclic ?f'*
proof (*rule kruskal-acyclic-inv*)
show *acyclic* (*f* ⊓ − *?e*^{T})

```
    proof −
      have f-intersect-below: (f ⊓ − ?e^T) ≤ f by simp
      have acyclic f
        by (meson assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def)
      thus ?thesis
        using comp-isotone dual-order.trans star-isotone f-intersect-below by blast
    qed
  next
    show covector ?p
      by (metis comp-associative vector-top-closed)
  next
    show (f ⊓ − ?e^T ⊓ ?p)^T * (f ⊓ − ?e^T)^⋆ * ?e = bot
    proof −
      have ?e ≤ − (f^T⋆ * f^⋆)
        by (simp add: assms(3))
      then have ?e * top * ?e ≤ − (f^T⋆ * f^⋆)
        by (metis arc-top-arc minarc-arc minarc-bot-iff semiring.mult-not-zero)
      then have ?e^T * top * ?e^T ≤ − (f^T⋆ * f^⋆)^T
       by (metis comp-associative conv-complement conv-dist-comp conv-isotone symmetric-top-closed)
      then have ?e^T * top * ?e^T ≤ − (f^T⋆ * f^⋆)
        by (simp add: conv-dist-comp conv-star-commute)
      then have ?e * (f^T⋆ * f^⋆) * ?e ≤ bot
        using triple-schroeder-p by auto
      then have 1: ?e * f^T⋆ * f^⋆ * ?e ≤ bot
        using mult-assoc by auto
      have 2: (f ⊓ − ?e^T)^T⋆ ≤ f^T⋆
        by (simp add: conv-dist-inf star-isotone)
      have (f ⊓ − ?e^T ⊓ ?p)^T * (f ⊓ − ?e^T)^⋆ * ?e ≤ (f ⊓ ?p)^T * (f ⊓ − ?e^T)^⋆ * ?e
        by (simp add: comp-isotone conv-dist-inf inf.orderI inf.sup-monoid.add-assoc)
      also have ... ≤ (f ⊓ ?p)^T * f^⋆ * ?e
        by (simp add: comp-isotone star-isotone)
      also have ... ≤ (f ⊓ top * ?e * (f)^T⋆)^T * f^⋆ * ?e
        using 2 by (metis comp-inf.comp-isotone comp-inf.coreflexive-transitive comp-isotone
conv-isotone inf.idem top.extremum)
      also have ... = (f^T ⊓ (top * ?e * f^T⋆)^T) * f^⋆ * ?e
        by (simp add: conv-dist-inf)
      also have ... ≤ top * (f^T ⊓ (top * ?e * f^T⋆)^T) * f^⋆ * ?e
        using top-left-mult-increasing mult-assoc by auto
      also have ... = (top ⊓ top * ?e * f^T⋆) * f^T * f^⋆ * ?e
        by (smt covector-comp-inf-1 covector-mult-closed eq-iff inf.sup-monoid.add-commute
vector-top-closed)
      also have ... = top * ?e * f^T⋆ * f^T * f^⋆ * ?e
        by simp
      also have ... ≤ top * ?e * f^T⋆ * f^⋆ * ?e
        by (smt conv-dist-comp conv-isotone conv-star-commute mult-left-isotone mult-right-isotone
star.left-plus-below-circ mult-assoc)
      also have ... ≤ bot
        using 1 covector-bot-closed le-bot mult-assoc by fastforce
      finally show ?thesis
        using le-bot by auto
    qed
  next
    show ?e * (f ⊓ − ?e^T)^⋆ * ?e = bot
    proof −
      have 1: ?e ≤ − ?F
        by (simp add: assms(3))
      have 2: injective f
        by (meson assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def)
```

```
      have 3: equivalence ?F
        using 2 forest-components-equivalence by simp
      then have 4: ?e^T = ?e^T * top * ?e^T
        using arc-conv-closed arc-top-arc covector-complement-closed covector-conv-vector ex231e
minarc-arc minarc-bot-iff pp-surjective regular-closed-top vector-mult-closed vector-top-closed by smt
      also have ... ≤ - ?F using 1 3 conv-isotone conv-complement calculation by fastforce
      finally have 5: ?e * ?F * ?e = bot
        using 4 triple-schroeder-p le-bot pp-total regular-closed-top vector-top-closed by smt
      have (f ⊓ - ?e^T)^⋆ ≤ f^⋆
        by (simp add: star-isotone)
      then have ?e * (f ⊓ - ?e^T)^⋆ * ?e ≤ ?e * f^⋆ * ?e
        using mult-left-isotone mult-right-isotone by blast
      also have ... ≤ ?e * ?F * ?e
        by (metis conv-star-commute forest-components-increasing mult-left-isotone mult-right-isotone
star-involutive)
      also have 6: ... = bot
        using 5 by simp
      finally show ?thesis using 6 le-bot by blast
    qed
  next
    show forest-components (f ⊓ -?e^T) ≤ - ?e
    proof -
      have 1: ?e ≤ - ?F
        by (simp add: assms(3))
      have f ⊓ - ?e^T ≤ f
        by simp
      then have forest-components (f ⊓ - ?e^T) ≤ ?F
        using forest-components-isotone by blast
      then show ?thesis
        using 1 order-lesseq-imp p-antitone-iff by blast
    qed
  qed
next
  show ?f' ≤ --g
  proof -
    have 1: (f ⊓ - ?e^T ⊓ - ?p) ≤ --g
      by (meson assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def inf.coboundedI1)
    have 2: (f ⊓ - ?e^T ⊓ ?p)^T ≤ --g
    proof -
      have (f ⊓ - ?e^T ⊓ ?p)^T ≤ f^T
        by (simp add: conv-isotone inf.sup-monoid.add-assoc)
      also have ... ≤ --g
        by (metis assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def conv-complement
conv-isotone)
      finally show ?thesis
        by simp
    qed
    have 3: ?e ≤ --g
      by (metis inf.boundedE minarc-below pp-dist-inf)
    show ?thesis using 1 2 3
      by simp
  qed
next
  show regular ?f'
    using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def minarc-regular
regular-closed-star regular-conv-closed regular-mult-closed by auto
next
  show ∃ w. minimum-spanning-forest w g ∧ ?f' ≤ w ⊔ w^T
```

proof (*rule exists-a-w*)
show *symmetric g*
using *assms*(*1*) *boruvka-inner-invariant-def boruvka-outer-invariant-def* **by** *auto*
next
show *forest f*
using *assms*(*1*) *boruvka-inner-invariant-def boruvka-outer-invariant-def* **by** *auto*
next
show $f \leq --g$
using *assms*(*1*) *boruvka-inner-invariant-def boruvka-outer-invariant-def* **by** *auto*
next
show *regular f*
using *assms*(*1*) *boruvka-inner-invariant-def boruvka-outer-invariant-def* **by** *auto*
next
show $(\exists\, w\ .\ \textit{minimum-spanning-forest}\ w\ g \wedge f \leq w \sqcup w^T)$
using *assms*(*1*) *boruvka-inner-invariant-def boruvka-outer-invariant-def* **by** *auto*
next
show *vector j*
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show *regular j*
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show *forest h*
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show *forest-components h* $\leq$ *forest-components f*
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show *big-forest* (*forest-components h*) *d*
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show $d * top \leq - j$
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show *forest-components h* $* j = j$
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show *forest-components f* $=$ (*forest-components h* $* (d \sqcup d^T))^\star *$ *forest-components h*
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show $f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show $(\forall\ a\ b\ .\ \textit{bf-between-arcs}\ a\ b\ (\textit{forest-components}\ h)\ d \wedge a \leq -(\textit{forest-components}\ h) \sqcap -- g \wedge b \leq d \longrightarrow \textit{sum}(b \sqcap g) \leq \textit{sum}(a \sqcap g))$
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show *regular d*
using *assms*(*1*) *boruvka-inner-invariant-def* **by** *blast*
next
show *selected-edge h j g* $\leq -$ *forest-components f*
by (*simp add*: *assms*(*3*))
next
show *selected-edge h j g* $\neq$ *bot*
by (*simp add*: *assms*(*4*))
next
show $j \neq bot$
by (*simp add*: *assms*(*2*))

```
    next
      show regular h
        using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
    next
      show h ≤ −−g
        using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
    qed
  qed
qed

lemma second-inner-invariant-when-e-not-bot:
  assumes boruvka-inner-invariant j f h g d
    and j ≠ bot
    and selected-edge h j g ≤ − forest-components f
    and selected-edge h j g ≠ bot
  shows boruvka-inner-invariant
        (j ⊓ − choose-component (forest-components h) j)
        (f ⊓ − selected-edge h j g^T ⊓ − path f h j g ⊔
         (f ⊓ − selected-edge h j g^T ⊓ path f h j g)^T ⊔
         selected-edge h j g)
        h g (d ⊔ selected-edge h j g)
proof −
  let ?c = choose-component (forest-components h) j
  let ?p = path f h j g
  let ?F = forest-components f
  let ?H = forest-components h
  let ?e = selected-edge h j g
  let ?f' = f ⊓ −?e^T ⊓ −?p ⊔ (f ⊓ −?e^T ⊓ ?p)^T ⊔ ?e
  let ?d' = d ⊔ ?e
  let ?j' = j ⊓ −?c
  show boruvka-inner-invariant ?j' ?f' h g ?d'
  proof (unfold boruvka-inner-invariant-def, intro conjI)
    have 1: boruvka-outer-invariant ?f' g
      using assms(1, 2, 3, 4) boruvka-outer-invariant-when-e-not-bot by blast
    show boruvka-outer-invariant ?f' g
      using assms(1, 2, 3, 4) boruvka-outer-invariant-when-e-not-bot by blast
    show g ≠ bot
      using assms(1) boruvka-inner-invariant-def by force
    show vector ?j'
      using assms(1, 2) boruvka-inner-invariant-def component-is-vector vector-complement-closed
vector-inf-closed by simp
    show regular ?j'
      using assms(1) boruvka-inner-invariant-def by auto
    show boruvka-outer-invariant h g
      by (meson assms(1) boruvka-inner-invariant-def)
    show injective h
      by (meson assms(1) boruvka-inner-invariant-def)
    show pd-kleene-allegory-class.acyclic h
      by (meson assms(1) boruvka-inner-invariant-def)
    show ?H ≤ forest-components ?f'
    proof −
      have 2: ?F ≤ forest-components ?f'
      proof (rule components-disj-increasing)
        show regular ?p
          using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def minarc-regular
regular-closed-star regular-conv-closed regular-mult-closed by auto[1]
      next
        show regular ?e
```

```
      using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def minarc-regular
regular-closed-star regular-conv-closed regular-mult-closed by auto[1]
    next
     show injective ?f'
      using 1 boruvka-outer-invariant-def by blast
    next
     show injective f
      using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by blast
    qed
    thus ?thesis
     using assms(1) boruvka-inner-invariant-def dual-order.trans by blast
   qed
   show big-forest ?H ?d'
    using assms(1, 2, 3, 4) big-forest-d-U-e boruvka-inner-invariant-def boruvka-outer-invariant-def
by auto
  next
   show ?d' * top ≤ -?j'
   proof -
    have 31: ?d' * top = d * top ⊔ ?e * top
     by (simp add: mult-right-dist-sup)
    have 32: d * top ≤ -?j'
     by (meson assms(1) boruvka-inner-invariant-def inf.coboundedI1 p-antitone-iff)
    have regular (?c * - ?c^T)
     using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def component-is-regular
regular-conv-closed regular-mult-closed by presburger
    then have minarc(?c * - ?c^T ⊓ g) = minarc(?c ⊓ - ?c^T ⊓ g)
     by (metis component-is-vector covector-comp-inf inf-top.left-neutral vector-conv-compl)
    also have ... ≤ -- (?c ⊓ - ?c^T ⊓ g)
     using minarc-below by blast
    also have ... ≤ -- ?c
     by (simp add: inf.sup-monoid.add-assoc)
    also have ... = ?c
     using component-is-regular by auto
    finally have ?e ≤ ?c
     by simp
    then have ?e * top ≤ ?c
     by (metis component-is-vector mult-left-isotone)
    also have ... ≤ -j ⊔ ?c
     by simp
    also have ... = - (j ⊓ - ?c)
     using component-is-regular by auto
    finally have 33: ?e * top ≤ - (j ⊓ - ?c)
     by simp
    show ?thesis
     using 31 32 33 by auto
   qed
  next
   show ?H * ?j' = ?j'
    using fc-j-eq-j-inv assms(1) boruvka-inner-invariant-def by blast
  next
   show forest-components ?f' = (?H * (?d' ⊔ ?d'^T))^* * ?H
   proof -
    have forest-components ?f' = (f ⊔ f^T ⊔ ?e ⊔ ?e^T)^*
    proof (rule simplify-forest-components-f)
     show regular ?p
      using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def minarc-regular
regular-closed-star regular-conv-closed regular-mult-closed by auto
    next
```

```
      show regular ?e
        using minarc-regular by auto
      next
        show injective ?f'
        using assms(1, 2, 3, 4) boruvka-outer-invariant-def boruvka-outer-invariant-when-e-not-bot by
blast
      next
        show injective f
          using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by blast
      qed
      also have ... = (h ⊔ h^T ⊔ d ⊔ d^T ⊔ ?e ⊔ ?e^T)^⋆
        using assms(1) boruvka-inner-invariant-def by simp
      also have ... = (h ⊔ h^T ⊔ ?d' ⊔ ?d'^T)^⋆
        by (smt conv-dist-sup sup-monoid.add-assoc sup-monoid.add-commute)
      also have ... = ((h ⊔ h^T)^⋆ ∗ (?d' ⊔ ?d'^T))^⋆ ∗ (h ⊔ h^T)^⋆
        by (metis star.circ-sup-9 sup-assoc)
      finally show ?thesis
        using assms(1) boruvka-inner-invariant-def forest-components-wcc by simp
    qed
  next
    show ?f' ⊔ ?f'^T = h ⊔ h^T ⊔ ?d' ⊔ ?d'^T
    proof −
      have ?f' ⊔ ?f'^T = f ⊓ − ?e^T ⊓ − ?p ⊔ (f ⊓ − ?e^T ⊓ ?p)^T ⊔ ?e ⊔ (f ⊓ − ?e^T ⊓ − ?p)^T ⊔ (f ⊓
− ?e^T ⊓ ?p) ⊔ ?e^T
        by (simp add: conv-dist-sup sup-monoid.add-assoc)
      also have ... = (f ⊓ − ?e^T ⊓ − ?p) ⊔ (f ⊓ − ?e^T ⊓ ?p) ⊔ (f ⊓ − ?e^T ⊓ ?p)^T ⊔ (f ⊓ − ?e^T ⊓
− ?p)^T ⊔ ?e^T ⊔ ?e
        by (simp add: sup.left-commute sup-commute)
      also have ... = f ⊔ f^T ⊔ ?e ⊔ ?e^T
      proof (rule simplify-f)
        show regular ?p
          using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def minarc-regular
regular-closed-star regular-conv-closed regular-mult-closed by auto
      next
        show regular ?e
          using minarc-regular by blast
      qed
      also have ... = h ⊔ h^T ⊔ d ⊔ d^T ⊔ ?e ⊔ ?e^T
        using assms(1) boruvka-inner-invariant-def by auto
      finally show ?thesis
        by (smt conv-dist-sup sup.left-commute sup-commute)
    qed
  next
    show ∀ a b . bf-between-arcs a b ?H ?d' ∧ a ≤ − ?H ⊓ −− g ∧ b ≤ ?d' ⟶ sum (b ⊓ g) ≤ sum
(a ⊓ g)
    proof (intro allI, rule impI, unfold bf-between-arcs-def)
      fix a b
      assume 1: (arc a ∧ arc b ∧ a^T ∗ top ≤ (?H ∗ ?d')^⋆ ∗ ?H ∗ b ∗ top) ∧ a ≤ − ?H ⊓ −− g ∧ b ≤
?d'
      thus sum (b ⊓ g) ≤ sum (a ⊓ g)
      proof (cases b = ?e)
        case b-equals-e: True
        thus ?thesis
        proof (cases a = ?e)
          case True
          thus ?thesis
            using b-equals-e by auto
        next
```

```
      case a-ne-e: False
      have sum (b ⊓ g) ≤ sum (a ⊓ g)
      proof (rule a-to-e-in-bigforest)
        show symmetric g
          using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
      next
        show j ≠ bot
          by (simp add: assms(2))
      next
        show f ≤ −− g
          using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
      next
        show vector j
          using assms(1) boruvka-inner-invariant-def by blast
      next
        show forest h
          using assms(1) boruvka-inner-invariant-def by blast
      next
        show  big-forest (forest-components h) d
          using assms(1) boruvka-inner-invariant-def by blast
      next
        show f ⊔ f^T = h ⊔ h^T ⊔ d ⊔ d^T
          using assms(1) boruvka-inner-invariant-def by blast
      next
        show ∀ a b. bf-between-arcs a b (?H) d ∧ a ≤ − ?H ⊓ − − g ∧ b ≤ d ⟶ sum (b ⊓ g) ≤
sum (a ⊓ g)
          using assms(1) boruvka-inner-invariant-def by blast
      next
        show regular d
          using assms(1) boruvka-inner-invariant-def by blast
      next
        show b = ?e
          using b-equals-e by simp
      next
        show arc a
          using 1 by simp
      next
        show bf-between-arcs a b ?H ?d'
          using 1 bf-between-arcs-def by simp
      next
        show a ≤ − ?H ⊓ −− g
          using 1 by simp
      next
        show regular h
          using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
      qed
      thus ?thesis
        by simp
    qed
  next
    case b-not-equal-e: False
    then have b-below-d: b ≤ d
      using 1 assms(4) different-arc-in-sup-arc minarc-arc minarc-bot-iff by metis
    thus ?thesis
    proof (cases ?e ≤ d)
      case True
      then have bf-between-arcs a b ?H d ∧ b ≤ d
        using 1 bf-between-arcs-def sup.absorb1 by auto
```

```
      thus ?thesis
        using 1 assms(1) boruvka-inner-invariant-def by blast
      next
        case e-not-less-than-d: False
        have 71:equivalence ?H
          using assms(1) fch-equivalence boruvka-inner-invariant-def by auto
        then have 72: bf-between-arcs a b ?H ?d' ⟷ bf-between-arcs a b ?H d ∨ (bf-between-arcs a
?e ?H d ∧ bf-between-arcs ?e b ?H d)
        proof (rule big-forest-path-split-disj)
          show arc ?e
            using assms(4) minarc-arc minarc-bot-iff by blast
        next
          show regular a ∧ regular b ∧ regular ?e ∧ regular d ∧ regular ?H
            using assms(1) 1 boruvka-inner-invariant-def boruvka-outer-invariant-def arc-regular
minarc-regular regular-closed-star regular-conv-closed regular-mult-closed by auto
        qed
        thus ?thesis
        proof (cases bf-between-arcs a b ?H d)
          case True
          have bf-between-arcs a b ?H d ∧ b ≤ d
            using 1 True bf-between-arcs-def sup.absorb1 by (metis assms(4) b-not-equal-e minarc-arc
minarc-bot-iff different-arc-in-sup-arc)
          thus ?thesis
            using 1 assms(1) b-below-d boruvka-inner-invariant-def by auto
        next
          case False
          have 73:bf-between-arcs a ?e ?H d ∧ bf-between-arcs ?e b ?H d
            using 1 72 False bf-between-arcs-def by blast
          have 74: ?e ≤ −−g
            by (metis inf.boundedE minarc-below pp-dist-inf)
          have ?e ≤ − ?H
            by (meson assms(1) assms(3) boruvka-inner-invariant-def dual-order.trans p-antitone-iff)
          then have ?e ≤ − ?H ⊓ −−g
            using 74 by simp
          then have 75: sum (b ⊓ g) ≤ sum (?e ⊓ g)
            using assms(1) b-below-d 73 boruvka-inner-invariant-def by blast
          have 76: bf-between-arcs a ?e ?H ?d'
            by (meson 73 big-forest-path-split-disj assms(1) bf-between-arcs-def
boruvka-inner-invariant-def boruvka-outer-invariant-def fch-equivalence arc-regular regular-closed-star
regular-conv-closed regular-mult-closed)
          have 77: sum (?e ⊓ g) ≤ sum (a ⊓ g)
          proof (rule a-to-e-in-bigforest)
            show symmetric g
              using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
          next
            show j ≠ bot
              by (simp add: assms(2))
          next
            show f ≤ −− g
              using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
          next
            show vector j
              using assms(1) boruvka-inner-invariant-def by blast
          next
            show forest h
              using assms(1) boruvka-inner-invariant-def by blast
          next
            show  big-forest (forest-components h) d
```

```
            using assms(1) boruvka-inner-invariant-def by blast
          next
            show f ⊔ f^T = h ⊔ h^T ⊔ d ⊔ d^T
              using assms(1) boruvka-inner-invariant-def by blast
          next
            show ∀ a b. bf-between-arcs a b (?H) d ∧ a ≤ - ?H ⊓ - - g ∧ b ≤ d ⟶ sum (b ⊓ g) ≤
sum (a ⊓ g)
              using assms(1) boruvka-inner-invariant-def by blast
          next
            show regular d
              using assms(1) boruvka-inner-invariant-def by blast
          next
            show ?e = ?e
              by simp
          next
            show arc a
              using 1 by simp
          next
            show bf-between-arcs a ?e ?H ?d'
              by (simp add: 76)
          next
            show a ≤ - ?H ⊓ --g
              using 1 by simp
          next
            show regular h
              using assms(1) boruvka-inner-invariant-def boruvka-outer-invariant-def by auto
          qed
          thus ?thesis
            using 75 order.trans by blast
        qed
      qed
    qed
  qed
  next
    show regular ?d'
      using assms(1) boruvka-inner-invariant-def minarc-regular by auto
  qed
qed

lemma second-inner-invariant-when-e-bot:
  assumes selected-edge h j g = bot
    and selected-edge h j g ≤ - forest-components f
    and boruvka-inner-invariant j f h g d
  shows boruvka-inner-invariant
     (j ⊓ - choose-component (forest-components h) j)
     (f ⊓ - selected-edge h j g^T ⊓ - path f h j g ⊔
      (f ⊓ - selected-edge h j g^T ⊓ path f h j g)^T ⊔
      selected-edge h j g)
     h g (d ⊔ selected-edge h j g)
proof -
  let ?c = choose-component (forest-components h) j
  let ?p = path f h j g
  let ?F = forest-components f
  let ?H = forest-components h
  let ?e = selected-edge h j g
  let ?f' = f ⊓ -?e^T ⊓ -?p ⊔ (f ⊓ -?e^T ⊓ ?p)^T ⊔ ?e
  let ?d' = d ⊔ ?e
  let ?j' = j ⊓ -?c
```

show *boruvka-inner-invariant ?j' ?f' h g ?d'*
proof (*unfold boruvka-inner-invariant-def*, *intro conjI*)
next
show *boruvka-outer-invariant ?f' g*
using *assms*(*1*) *assms*(*3*) *boruvka-inner-invariant-def* **by** *auto*
next
show $g \neq bot$
using *assms*(*3*) *boruvka-inner-invariant-def* **by** *blast*
next
show *vector ?j'*
by (*metis assms*(*3*) *boruvka-inner-invariant-def component-is-vector vector-complement-closed vector-inf-closed*)
next
show *regular ?j'*
using *assms*(*3*) *boruvka-inner-invariant-def* **by** *auto*
next
show *boruvka-outer-invariant h g*
using *assms*(*3*) *boruvka-inner-invariant-def* **by** *blast*
next
show *injective h*
using *assms*(*3*) *boruvka-inner-invariant-def* **by** *blast*
next
show *pd-kleene-allegory-class.acyclic h*
using *assms*(*3*) *boruvka-inner-invariant-def* **by** *blast*
next
show $?H \leq$ *forest-components ?f'*
using *assms*(*1*) *assms*(*3*) *boruvka-inner-invariant-def* **by** *auto*
next
show *big-forest ?H ?d'*
using *assms*(*1*) *assms*(*3*) *boruvka-inner-invariant-def* **by** *auto*
next
show $?d' * top \leq -?j'$
using *assms*(*1*) *assms*(*3*) *boruvka-inner-invariant-def* **by** (*metis order.trans p-antitone-inf sup-monoid.add-0-right*)
next
show $?H * ?j' = ?j'$
using *assms*(*3*) *fc-j-eq-j-inv boruvka-inner-invariant-def* **by** *blast*
next
show $forest\text{-}components\ ?f' = (?H * (?d' \sqcup ?d'^{T}))^{\star} * ?H$
using *assms*(*1*, *3*) *boruvka-inner-invariant-def* **by** *auto*
next
show $?f' \sqcup ?f'^{T} = h \sqcup h^{T} \sqcup ?d' \sqcup ?d'^{T}$
using *assms*(*1*, *3*) *boruvka-inner-invariant-def* **by** *auto*
next
show $\forall a\ b.\ bf\text{-}between\text{-}arcs\ a\ b\ ?H\ ?d' \wedge a \leq -?H \sqcap --g \wedge b \leq ?d' \longrightarrow sum(b \sqcap g) \leq sum(a \sqcap g)$
using *assms*(*1*, *3*) *boruvka-inner-invariant-def* **by** *auto*
next
show *regular ?d'*
using *assms*(*1*) *assms*(*3*) *boruvka-inner-invariant-def* **by** *auto*
qed
qed

B.2.4 Formalization and proof of Borůvka's minimum spanning tree algorithm

The following result shows that Borůvka's algorithm constructs a minimum spanning forest. We have the same postcondition as Guttmann's proof of Kruskal's minimum spanning tree

algorithm. We show only partial correctness.

```
theorem boruvka-mst:
  VARS f j h c e d
  { symmetric g }
  f := bot;
  WHILE −(forest-components f) ⊓ g ≠ bot
    INV { boruvka-outer-invariant f g }
    DO
      j := top;
      h := f;
      d := bot;
      WHILE j ≠ bot
        INV { boruvka-inner-invariant j f h g d }
        DO
          c := choose-component (forest-components h) j;
          e := minarc(c ∗ −c^T ⊓ g);
          IF e ≤ −(forest-components f) THEN
            f := f ⊓ −e^T;
            f := (f ⊓ −(top ∗ e ∗ f^T⋆)) ⊔ (f ⊓ top ∗ e ∗ f^T⋆)^T ⊔ e;
            d := d ⊔ e
          ELSE
            SKIP
          FI;
          j := j ⊓ −c
        OD
    OD
  { minimum-spanning-forest f g }
proof vcg-simp
  assume 1: symmetric g
  show boruvka-outer-invariant bot g
    using 1 boruvka-outer-invariant-def kruskal-exists-minimal-spanning by auto
next
  fix f
  let ?F = forest-components f
  assume 1: boruvka-outer-invariant f g ∧ − ?F ⊓ g ≠ bot
  have 2: equivalence ?F
    using 1 boruvka-outer-invariant-def forest-components-equivalence by auto
  show boruvka-inner-invariant top f f g bot
  proof (unfold boruvka-inner-invariant-def, intro conjI)
    show boruvka-outer-invariant f g
      by (simp add: 1)
  next
    show g ≠ bot
      using 1 by auto
  next
    show surjective top
      by simp
  next
    show regular top
      by simp
  next
    show boruvka-outer-invariant f g
      using 1 by auto
  next
    show injective f
      using 1 boruvka-outer-invariant-def by blast
  next
    show pd-kleene-allegory-class.acyclic f
```

```
    using 1 boruvka-outer-invariant-def by blast
  next
    show ?F ≤ ?F
      by simp
  next
    show big-forest ?F bot
      by (simp add: 2 big-forest-def)
  next
    show bot * top ≤ - top
      by simp
  next
    show times-top-class.total (?F)
      by (simp add: star.circ-right-top mult-assoc)
  next
    show ?F = (?F * (bot ⊔ bot^T))^⋆ * ?F
      by (metis mult-right-zero semiring.mult-zero-left star.circ-loop-fixpoint sup-commute
sup-monoid.add-0-right symmetric-bot-closed)
  next
    show f ⊔ f^T = f ⊔ f^T ⊔ bot ⊔ bot^T
      by simp
  next
    show ∀ a b. bf-between-arcs a b ?F bot ∧ a ≤ - ?F ⊓ -- g ∧ b ≤ bot ⟶ sum (b ⊓ g) ≤ sum (a
⊓ g)
      by (metis (full-types) bf-between-arcs-def bot-unique mult-left-zero mult-right-zero top.extremum)
  next
    show regular bot
      by auto
  qed
next
  fix f j h d
  let ?c = choose-component (forest-components h) j
  let ?p = path f h j g
  let ?F = forest-components f
  let ?H = forest-components h
  let ?e = selected-edge h j g
  let ?f' = f ⊓ -?e^T ⊓ -?p ⊔ (f ⊓ -?e^T ⊓ ?p)^T ⊔ ?e
  let ?d' = d ⊔ ?e
  let ?j' = j ⊓ -?c
  assume 1: boruvka-inner-invariant j f h g d ∧ j ≠ bot
  show (?e ≤ -?F ⟶ boruvka-inner-invariant ?j' ?f' h g ?d') ∧ (¬ ?e ≤ -?F ⟶
boruvka-inner-invariant ?j' f h g d)
  proof (intro conjI)
    show ?e ≤ -?F ⟶ boruvka-inner-invariant ?j' ?f' h g ?d'
    proof (cases ?e = bot)
      case True
      then show ?thesis
        using 1 second-inner-invariant-when-e-bot by simp
    next
      case False
      then show ?thesis
        using 1 second-inner-invariant-when-e-not-bot by simp
    qed
  next
    show ¬ ?e ≤ -?F ⟶ boruvka-inner-invariant ?j' f h g d
    proof (rule impI, unfold boruvka-inner-invariant-def, intro conjI)
      show boruvka-outer-invariant f g
        using 1 boruvka-inner-invariant-def by blast
    next
```

```
    show g ≠ bot
      using 1 boruvka-inner-invariant-def by blast
  next
    show vector ?j'
      using 1 boruvka-inner-invariant-def component-is-vector vector-complement-closed
vector-inf-closed by auto
  next
    show regular ?j'
      using 1 boruvka-inner-invariant-def by auto
  next
    show boruvka-outer-invariant h g
      using 1 boruvka-inner-invariant-def by auto
  next
    show injective h
      using 1 boruvka-inner-invariant-def by blast
  next
    show pd-kleene-allegory-class.acyclic h
      using 1 boruvka-inner-invariant-def by blast
  next
    show ?H ≤ ?F
      using 1 boruvka-inner-invariant-def by blast
  next
    show big-forest ?H d
      using 1 boruvka-inner-invariant-def by blast
  next
    show d * top ≤ -?j'
      using 1 boruvka-inner-invariant-def by (meson dual-order.trans p-antitone-inf)
  next
    show ?H * ?j' = ?j'
      using 1 fc-j-eq-j-inv boruvka-inner-invariant-def by blast
  next
    show ?F = (?H * (d ⊔ d^T))^⋆ * ?H
      using 1 boruvka-inner-invariant-def by blast
  next
    show f ⊔ f^T = h ⊔ h^T ⊔ d ⊔ d^T
      using 1 boruvka-inner-invariant-def by blast
  next
    show ¬ ?e ≤ -?F ⟹ ∀ a b. bf-between-arcs a b ?H d ∧ a ≤ -?H ⊓ --g ∧ b ≤ d ⟶ sum(b ⊓
g) ≤ sum(a ⊓ g)
      using 1 boruvka-inner-invariant-def by blast
  next
    show ¬ ?e ≤ -?F ⟹ regular d
      using 1 boruvka-inner-invariant-def by blast
  qed
 qed
next
 fix f j h d
 assume 1: boruvka-inner-invariant j f h g d ∧ j = bot
 then show boruvka-outer-invariant f g
   by (meson 1 boruvka-inner-invariant-def)
next
 fix f
 assume 1: boruvka-outer-invariant f g ∧ - forest-components f ⊓ g = bot
 then have 2:spanning-forest f g
 proof (unfold spanning-forest-def, intro conjI)
   show injective f
     using 1 boruvka-outer-invariant-def by blast
 next
```

```
    show acyclic f
      using 1 boruvka-outer-invariant-def by blast
  next
    show f ≤ −−g
      using 1 boruvka-outer-invariant-def by blast
  next
    show components g ≤ forest-components f
    proof −
      let ?F = forest-components f
      have −?F ⊓ g ≤ bot
        by (simp add: 1)
      then have −−g ≤ bot ⊔ −−?F
        using 1 shunting-p p-antitone pseudo-complement by auto
      then have −−g ≤ ?F
        using 1 boruvka-outer-invariant-def pp-dist-comp pp-dist-star regular-conv-closed by auto
      then have (−−g)⋆ ≤ ?F⋆
        by (simp add: star-isotone)
      thus ?thesis
        using 1 boruvka-outer-invariant-def forest-components-star by auto
    qed
  next
    show regular f
      using 1 boruvka-outer-invariant-def by auto
  qed
  from 1 obtain w where 3: minimum-spanning-forest w g ∧ f ≤ w ⊔ w^T
    using boruvka-outer-invariant-def by blast
  hence w = w ⊓ −−g
    by (simp add: inf.absorb1 minimum-spanning-forest-def spanning-forest-def)
  also have ... ≤ w ⊓ components g
    by (metis inf.sup-right-isotone star.circ-increasing)
  also have ... ≤ w ⊓ f^T⋆ ∗ f⋆
    using 2 spanning-forest-def inf.sup-right-isotone by simp
  also have ... ≤ f ⊔ f^T
  proof (rule cancel-separate-6[where z=w and y=w^T])
    show injective w
      using 3 minimum-spanning-forest-def spanning-forest-def by simp
  next
    show f^T ≤ w^T ⊔ w
      using 3 by (metis conv-dist-inf conv-dist-sup conv-involutive inf.cobounded2 inf.orderE)
  next
    show f ≤ w^T ⊔ w
      using 3 by (simp add: sup-commute)
  next
    show injective w
      using 3 minimum-spanning-forest-def spanning-forest-def by simp
  next
    show w ⊓ w^T⋆ = bot
      using 3 by (metis acyclic-star-below-complement comp-inf.mult-right-isotone inf-p le-bot
minimum-spanning-forest-def spanning-forest-def)
  qed
  finally have 4: w ≤ f ⊔ f^T
    by simp
  have sum (f ⊓ g) = sum ((w ⊔ w^T) ⊓ (f ⊓ g))
    using 3 by (metis inf-absorb2 inf.assoc)
  also have ... = sum (w ⊓ (f ⊓ g)) + sum (w^T ⊓ (f ⊓ g))
    using 3 inf.commute acyclic-asymmetric sum-disjoint minimum-spanning-forest-def
spanning-forest-def by simp
  also have ... = sum (w ⊓ (f ⊓ g)) + sum (w ⊓ (f^T ⊓ g^T))
```

```
    by (metis conv-dist-inf conv-involutive sum-conv)
  also have ... = sum (f ⊓ (w ⊓ g)) + sum (f^T ⊓ (w ⊓ g))
  proof −
    have 51:f^T ⊓ (w ⊓ g) = f^T ⊓ (w ⊓ g^T)
      using 1 boruvka-outer-invariant-def by auto
    have 52:f ⊓ (w ⊓ g) = w ⊓ (f ⊓ g)
      by (simp add: inf.left-commute)
    thus ?thesis
      using 51 52 abel-semigroup.left-commute inf.abel-semigroup-axioms by fastforce
  qed
  also have ... = sum ((f ⊔ f^T) ⊓ (w ⊓ g))
    using 2 acyclic-asymmetric inf.sup-monoid.add-commute sum-disjoint spanning-forest-def by simp
  also have ... = sum (w ⊓ g)
    using 4 by (metis inf-absorb2 inf.assoc)
  finally show minimum-spanning-forest f g
    using 2 3 minimum-spanning-forest-def by simp
qed

end

end
```