Relational Minimum Spanning Tree Algorithms

Walter Guttmann and Nicolas Robinson-O'Brien

December 14, 2021

Abstract

We verify the correctness of Prim's, Kruskal's and Borůvka's minimum spanning tree algorithms based on algebras for aggregation and minimisation.

Contents

1 Overview

The theories described in this document prove the correctness of Prim's, Kruskal's and Borůvka's minimum spanning tree algorithms. Specifications and algorithms work in Stone-Kleene relation algebras extended by operations for aggregation and minimisation. The algorithms are implemented in a simple imperative language and their proof uses Hoare logic. The correctness proofs are discussed in [3, 5, 6, 8].

1.1 Prim's and Kruskal's minimum spanning tree algorithms

A framework based on Stone relation algebras and Kleene algebras and extended by operations for aggregation and minimisation was presented by the first author in [3, 5] and used to formally verify the correctness of Prim's minimum spanning tree algorithm. It was extended in [6] and applied to prove the correctness of Kruskal's minimum spanning tree algorithm.

Two theories, one each for Prim's and Kruskal's algorithms, prove total correctness of these algorithms. As case studies for the algebraic framework, these two theories combined were originally part of another AFP entry [4].

1.2 Borůvka's minimum spanning tree algorithm

Otakar Borůvka formalised the minimum spanning tree problem and proposed a solution to it [1]. Borůvka's original paper is written in Czech; translations of varying completeness can be found in [2, 7].

The theory for Borůvka's minimum spanning tree algorithm proves partial correctness of this algorithm. This work is based on the same algebraic framework as the proof of Kruskal's algorithm; in particular it uses many theories from the hierarchy underlying [4].

The theory for Borůvka's algorithm formally verifies results from the second author's Master's thesis [8].

2 Kruskal's Minimum Spanning Tree Algorithm

In this theory we prove total correctness of Kruskal's minimum spanning tree algorithm. The proof uses the following steps [6]. We first establish that the algorithm terminates and constructs a spanning tree. This is a constructive proof of the existence of a spanning tree; any spanning tree algorithm could be used for this. We then conclude that a minimum spanning tree exists. This is necessary to establish the invariant for the actual correctness proof, which shows that Kruskal's algorithm produces a minimum spanning tree.

theory *Kruskal*

imports *HOL−Hoare.Hoare-Logic Aggregation-Algebras.Aggregation-Algebras*

begin

context *m-kleene-algebra*
begin

definition *spanning-forest f g* ≡ *forest f* ∧ *f* ≤ −−*g* ∧ *components g* ≤ *forest-components f* ∧ *regular f*
definition *minimum-spanning-forest f g* ≡ *spanning-forest f g* ∧ (∀ *u* . *spanning-forest u g* ⟶ *sum* (*f* ⊓ *g*) ≤ *sum* (*u* ⊓ *g*))

definition *kruskal-spanning-invariant f g h* ≡ *symmetric g* ∧ *h* = h^T ∧ *g* ⊓ −−*h* = *h* ∧ *spanning-forest f* (−*h* ⊓ *g*)
definition *kruskal-invariant f g h* ≡ *kruskal-spanning-invariant f g h* ∧ (∃ *w* . *minimum-spanning-forest w g* ∧ *f* ≤ *w* ⊔ w^T)

We first show two verification conditions which are used in both correctness proofs.

lemma *kruskal-vc-1*:
assumes *symmetric g*
shows *kruskal-spanning-invariant bot g g*
⟨*proof*⟩

lemma *kruskal-vc-2*:
assumes *kruskal-spanning-invariant f g h*
and *h* ≠ *bot*
shows (*minarc h* ≤ −*forest-components f* ⟶ *kruskal-spanning-invariant* ((*f* ⊓ −(*top* ∗ *minarc h* ∗ $f^{T\star}$)) ⊔ (*f* ⊓ *top* ∗ *minarc h* ∗ $f^{T\star}$)^T ⊔ *minarc h*) *g* (*h* ⊓ −*minarc h* ⊓ −*minarc* h^T)
∧ *card* { *x* . *regular x* ∧ *x* ≤ −−*h* ∧ *x* ≤ −*minarc h* ∧ *x* ≤ −*minarc* h^T } < *card* { *x* . *regular x* ∧ *x* ≤ −−*h* }) ∧
(¬ *minarc h* ≤ −*forest-components f* ⟶ *kruskal-spanning-invariant f g* (*h* ⊓ −*minarc h* ⊓ −*minarc* h^T)
∧ *card* { *x* . *regular x* ∧ *x* ≤ −−*h* ∧ *x* ≤ −*minarc h* ∧ *x* ≤ −*minarc* h^T } < *card* { *x* . *regular x* ∧ *x* ≤ −−*h* })
⟨*proof*⟩

The following result shows that Kruskal's algorithm terminates and constructs a spanning tree. We cannot yet show that this is a minimum spanning tree.

theorem *kruskal-spanning*:
VARS e f h
[*symmetric g*]
f := *bot*;
h := *g*;
WHILE h ≠ *bot*
INV { *kruskal-spanning-invariant f g h* }
VAR { *card* { *x* . *regular x* ∧ *x* ≤ −−*h* } }
DO e := *minarc h*;
IF e ≤ −*forest-components f THEN*
f := (*f* ⊓ −(*top* ∗ *e* ∗ $f^{T\star}$)) ⊔ (*f* ⊓ *top* ∗ *e* ∗ $f^{T\star}$)^T ⊔ *e*
ELSE
SKIP
FI;
h := *h* ⊓ −*e* ⊓ −e^T
OD
[*spanning-forest f g*]
⟨*proof*⟩

Because we have shown total correctness, we conclude that a spanning tree exists.

lemma *kruskal-exists-spanning*:
symmetric g $\Longrightarrow \exists f$. *spanning-forest f g*
⟨*proof*⟩

This implies that a minimum spanning tree exists, which is used in the subsequent correctness proof.

lemma *kruskal-exists-minimal-spanning*:
assumes *symmetric g*
shows $\exists f$. *minimum-spanning-forest f g*
⟨*proof*⟩

Kruskal's minimum spanning tree algorithm terminates and is correct. This is the same algorithm that is used in the previous correctness proof, with the same precondition and variant, but with a different invariant and postcondition.

theorem *kruskal*:
VARS e f h
[*symmetric g*]
$f := bot$;
$h := g$;
WHILE $h \neq bot$
INV { *kruskal-invariant f g h* }
VAR { *card* { x . *regular* $x \wedge x \leq --h$ } }
DO $e :=$ *minarc h*;
IF $e \leq -$*forest-components f THEN*
$f := (f \sqcap -(top * e * f^{T\star})) \sqcup (f \sqcap top * e * f^{T\star})^T \sqcup e$
ELSE
SKIP
FI;
$h := h \sqcap -e \sqcap -e^T$
OD
[*minimum-spanning-forest f g*]
⟨*proof*⟩

end

end

3 Prim's Minimum Spanning Tree Algorithm

In this theory we prove total correctness of Prim's minimum spanning tree algorithm. The proof has the same overall structure as the total-correctness proof of Kruskal's algorithm [6]. The partial-correctness proof of Prim's algorithm is discussed in [3, 5].

theory *Prim*

imports *HOL−Hoare.Hoare-Logic Aggregation-Algebras.Aggregation-Algebras*

begin

context *m-kleene-algebra*
begin

abbreviation *component g r* ≡ $r^T * (--g)^\star$
definition *spanning-tree t g r* ≡ *forest t* ∧ $t \leq (component\ g\ r)^T * (component\ g\ r) \sqcap --g \wedge component\ g\ r \leq r^T * t^\star \wedge regular\ t$
definition *minimum-spanning-tree t g r* ≡ *spanning-tree t g r* ∧ (∀ *u* . *spanning-tree u g r* ⟶ $sum\ (t \sqcap g) \leq sum\ (u \sqcap g)$)
definition *prim-precondition g r* ≡ $g = g^T \wedge injective\ r \wedge vector\ r \wedge regular\ r$
definition *prim-spanning-invariant t v g r* ≡ *prim-precondition g r* ∧ $v^T = r^T * t^\star \wedge spanning\text{-}tree\ t\ (v * v^T \sqcap g)\ r$
definition *prim-invariant t v g r* ≡ *prim-spanning-invariant t v g r* ∧ (∃ *w* . *minimum-spanning-tree w g r* ∧ *t* ≤ *w*)

lemma *span-tree-split*:
assumes *vector r*
shows $t \leq (component\ g\ r)^T * (component\ g\ r) \sqcap --g \longleftrightarrow (t \leq (component\ g\ r)^T \wedge t \leq component\ g\ r \wedge t \leq --g)$
⟨*proof*⟩

lemma *span-tree-component*:
assumes *spanning-tree t g r*
shows *component g r* = *component t r*
⟨*proof*⟩

We first show three verification conditions which are used in both correctness proofs.

lemma *prim-vc-1*:
assumes *prim-precondition g r*
shows *prim-spanning-invariant bot r g r*
⟨*proof*⟩

lemma *prim-vc-2*:
assumes *prim-spanning-invariant t v g r*
and $v * -v^T \sqcap g \neq bot$
shows $prim\text{-}spanning\text{-}invariant\ (t \sqcup minarc\ (v * -v^T \sqcap g))\ (v \sqcup minarc\ (v * -v^T \sqcap g)^T * top)\ g\ r \wedge card\ \{\ x\ .\ regular\ x \wedge x \leq component\ g\ r \wedge x \leq -(v \sqcup minarc\ (v * -v^T \sqcap g)^T * top)^T\ \} < card\ \{\ x\ .\ regular\ x \wedge x \leq component\ g\ r \wedge x \leq -v^T\ \}$
⟨*proof*⟩

lemma *prim-vc-3*:
assumes *prim-spanning-invariant t v g r*
and $v * -v^T \sqcap g = bot$
shows *spanning-tree t g r*
⟨*proof*⟩

The following result shows that Prim's algorithm terminates and constructs a spanning tree. We cannot yet show that this is a minimum spanning tree.

theorem *prim-spanning*:
VARS t v e
[*prim-precondition g r*]
$t := bot$;
$v := r$;
WHILE $v * -v^T \sqcap g \neq bot$
INV { *prim-spanning-invariant t v g r* }
VAR { *card* { x . *regular* $x \wedge x \leq$ *component g r* $\sqcap -v^T$ } }
DO $e :=$ *minarc* $(v * -v^T \sqcap g)$;
$t := t \sqcup e$;
$v := v \sqcup e^T * top$
OD
[*spanning-tree t g r*]
⟨*proof*⟩

Because we have shown total correctness, we conclude that a spanning tree exists.

lemma *prim-exists-spanning*:
prim-precondition g r $\Longrightarrow \exists t$. *spanning-tree t g r*
⟨*proof*⟩

This implies that a minimum spanning tree exists, which is used in the subsequent correctness proof.

lemma *prim-exists-minimal-spanning*:
assumes *prim-precondition g r*
shows $\exists t$. *minimum-spanning-tree t g r*
⟨*proof*⟩

Prim's minimum spanning tree algorithm terminates and is correct. This is the same algorithm that is used in the previous correctness proof, with the same precondition and variant, but with a different invariant and postcondition.

theorem *prim*:
VARS t v e
[*prim-precondition g r* $\wedge$ $(\exists w$. *minimum-spanning-tree w g r*$)$]
$t := bot$;
$v := r$;
WHILE $v * -v^T \sqcap g \neq bot$
INV { *prim-invariant t v g r* }
VAR { *card* { x . *regular* $x \wedge x \leq$ *component g r* $\sqcap -v^T$ } }
DO $e :=$ *minarc* $(v * -v^T \sqcap g)$;
$t := t \sqcup e$;
$v := v \sqcup e^T * top$
OD
[*minimum-spanning-tree t g r*]

⟨*proof*⟩

end

end

4 Borůvka's Minimum Spanning Tree Algorithm

In this theory we prove partial correctness of Borůvka's minimum spanning tree algorithm.

theory *Boruvka*

imports
Relational-Disjoint-Set-Forests.Disjoint-Set-Forests
Kruskal

begin

4.1 General results

The proof is carried out in *m-k*-Stone-Kleene relation algebras. In this section we give results that hold more generally.

context *stone-kleene-relation-algebra*
begin

lemma *He-eq-He-THe-star*:
assumes *arc e*
and *equivalence H*
shows $H * e = H * e * (top * H * e)^\star$
⟨*proof*⟩

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*⟩

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*⟩

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*⟩

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*⟩

lemma *fch-equivalence*:
assumes *forest h*
shows *equivalence* (*forest-components h*)
⟨*proof*⟩

lemma *forest-modulo-equivalence-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*⟩

lemma *dTransHd-le-1*:
assumes *equivalence H*
and *univalent* $(H * d)$
shows $d^T * H * d \leq 1$
⟨*proof*⟩

lemma *HcompaT-le-compHaT*:
assumes *equivalence H*
and *injective* $(a * top)$
shows $-H * a * top \leq - (H * a * top)$
⟨*proof*⟩

4.2 Forests modulo an equivalence

In the graphical interpretation, the arcs of d are directed towards the root(s) of the *forest-modulo-equivalence*.

definition *forest-modulo-equivalence x d* $\equiv$ *equivalence x* $\wedge$ *univalent* $(x * d) \wedge x \sqcap d * d^T \leq 1 \wedge (x * d)^+ \sqcap x \leq bot$

definition *forest-modulo-equivalence-path a b H d* $\equiv$ *arc a* $\wedge$ *arc b* $\wedge$ $a^T * top \leq (H * d)^\star * H * b * top$

lemma *d-separates-forest-modulo-equivalence-1*:
assumes *forest-modulo-equivalence x d*

shows $x * d \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-2*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow d * x \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-3*:
assumes *forest-modulo-equivalence* $x\ d$
shows $d \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-4*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow d^T \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-5*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow d \sqcup d^T \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-6*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow d * x \sqcup x * d \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-7*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow x * (d \sqcup d^T) * x \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-8*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow (d * x)^T \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-9*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow (x * d)^T \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-10*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow (d * x)^+ \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-11*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow (x * d)^+ \leq -x$
⟨*proof*⟩

lemma *d-separates-forest-modulo-equivalence-12*:
shows *forest-modulo-equivalence* $x\ d \Longrightarrow (d * x)^{T+} \leq -x$
⟨*proof*⟩

lemma *d-separates-x-in-forest-13*:

shows *forest-modulo-equivalence x d* $\Longrightarrow$ $(x * d)^{T+} \leq -x$
⟨*proof*⟩

lemma *irreflexive-d-in-forest-modulo-equivalence*:
shows *forest-modulo-equivalence x d* $\Longrightarrow$ *irreflexive d*
⟨*proof*⟩

lemma *univalent-d-in-forest-modulo-equivalence*:
assumes *forest-modulo-equivalence x d*
shows *univalent d*
⟨*proof*⟩

lemma *acyclic-d-in-forest-modulo-equivalence*:
assumes *forest-modulo-equivalence x d*
shows *acyclic d*
⟨*proof*⟩

lemma *acyclic-dt-d-in-forest-modulo-equivalence*:
shows *forest-modulo-equivalence x d* $\Longrightarrow$ *acyclic* (d^T)
⟨*proof*⟩

lemma *dt-forest-modulo-equivalence-forest*:
shows *forest-modulo-equivalence x d* $\Longrightarrow$ *forest* (d^T)
⟨*proof*⟩

lemma *var-forest-modulo-equivalence-axiom*:
shows *forest-modulo-equivalence x d* $\Longrightarrow$ $d^T * x * d \leq 1$
⟨*proof*⟩

lemma *forest-modulo-equivalence-wcc*:
assumes *forest-modulo-equivalence x d*
shows $(x * d)^\star * (x * d)^{T\star} = ((x * d) \sqcup (x * d)^T)^\star$
⟨*proof*⟩

lemma *forest-modulo-equivalence-direction-1*:
assumes *forest-modulo-equivalence x d*
shows $(x * d)^\star \sqcap (x * d)^T = bot$
⟨*proof*⟩

lemma *forest-modulo-equivalence-direction-2*:
assumes *forest-modulo-equivalence x d*
shows $(x * d)^{T\star} \sqcap (x * d) \leq bot$
⟨*proof*⟩

lemma *forest-modulo-equivalence-separate*:
assumes *forest-modulo-equivalence x d*
shows $(x * d)^\star * (x * d)^{T\star} \sqcap (x * d)^T * (x * d) \leq 1$
⟨*proof*⟩

lemma *forest-modulo-equivalence-path-trans-closure*:
assumes *forest-modulo-equivalence x d*
shows $(x * d^T)^+ * x * d * x * d^T \leq (x * d^T)^+$
⟨*proof*⟩

The *forest-modulo-equivalence* structure is preserved if d is decreased.

lemma *forest-modulo-equivalence-decrease-d*:
assumes *forest-modulo-equivalence x d*
shows *forest-modulo-equivalence x* $(d \sqcap c)$
⟨*proof*⟩

lemma *expand-forest-modulo-equivalence*:
assumes *forest-modulo-equivalence H d*
shows $(d^T * H)^\star * (H * d)^\star = (d^T * H)^\star \sqcup (H * d)^\star$
⟨*proof*⟩

lemma *forest-modulo-equivalence-path-bot*:
assumes *arc a*
and $a \leq d$
and *forest-modulo-equivalence H d*
shows $(d \sqcap - a)^T * (H * a * top) \leq bot$
⟨*proof*⟩

lemma *forest-modulo-equivalence-path-split-2*:
assumes *arc a*
and $a \leq d$
and *forest-modulo-equivalence H d*
shows $(H * (d \sqcap - a))^\star * H * a * top = (H * ((d \sqcap - a) \sqcup (d \sqcap - a)^T))^\star * H * a * top$
⟨*proof*⟩

end

context *stone-relation-algebra*
begin

A *vector-classes* corresponds to one or more equivalence classes and a *unique-vector-class* corresponds to a single equivalence class.

definition *vector-classes* :: $'a \Rightarrow 'a \Rightarrow bool$ **where** *vector-classes x v* $\equiv$ *regular x* $\wedge$ *regular v* $\wedge$ *equivalence x* $\wedge$ *vector v* $\wedge$ $x * v \leq v \wedge v \neq bot$
definition *unique-vector-class* :: $'a \Rightarrow 'a \Rightarrow bool$ **where** *unique-vector-class x v* $\equiv$ *vector-classes x v* $\wedge$ $v * v^T \leq x$

end

4.3 An operation to select components

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.
* Finally, axiom *component-not-bot-when-v-bot-bot* expresses that the operation *choose-component* returns a non-empty component if the input satisfies the given criteria.

class *choose-component* =
  fixes *choose-component* :: $'a \Rightarrow 'a \Rightarrow 'a$

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

Every *m-kleene-algebra* is an instance of *choose-component-algebra* when the *choose-component* operation is defined as follows:

context *m-kleene-algebra*
begin

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

sublocale *m-choose-component-algebra*: *choose-component-algebra* **where**
choose-component = *m-choose-component*
⟨*proof*⟩

end

4.4 m-k-Stone-Kleene relation algebras

m-k-Stone-Kleene relation algebras are an extension of *m*-Kleene algebras where the *choose-component* operation has been added.

class *m-kleene-algebra-choose-component* =
m-kleene-algebra
+ *choose-component-algebra*
begin

A *selected-edge* is a minimum-weight edge whose source is in a component, with respect to h, j and g, and whose target is not in that component.

abbreviation *selected-edge* $h\ j\ g \equiv$ *minarc* (*choose-component* (*forest-components* h) $j * -$ *choose-component* (*forest-components* h) $j^T \sqcap g$)

A *path* is any sequence of edges in the forest, f, of the graph, g, backwards from the target of the *selected-edge* to a root in f.

abbreviation *path* $f\ h\ j\ g \equiv$ *top* $*$ *selected-edge* $h\ j\ g * (f \sqcap -$ *selected-edge* $h\ j\ g^T)^{T\star}$

definition *boruvka-outer-invariant* $f\ g \equiv$
symmetric g
$\wedge$ *forest* f
$\wedge\ f \leq --g$
$\wedge$ *regular* f
$\wedge\ (\exists\, w\ .$ *minimum-spanning-forest* $w\ g \wedge f \leq w \sqcup w^T)$

definition *boruvka-inner-invariant* $j\ f\ h\ g\ d \equiv$
boruvka-outer-invariant $f\ g$
$\wedge\ g \neq$ *bot*
$\wedge$ *regular* d
$\wedge$ *regular* $j \wedge$ *vector* j
$\wedge$ *regular* $h \wedge$ *forest* h
$\wedge$ *forest-components* $h * j = j$
$\wedge$ *forest-modulo-equivalence* (*forest-components* h) d
$\wedge\ d * top \leq - j$
$\wedge\ f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$
$\wedge\ (\forall\ a\ b\ .$ *forest-modulo-equivalence-path* $a\ b$ (*forest-components* h) $d \wedge a \leq -$(*forest-components* h) $\sqcap --\ g \wedge b \leq d \longrightarrow$ *sum*$(b \sqcap g) \leq$ *sum*$(a \sqcap g)$)

lemma *F-is-H-and-d*:
assumes $f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$
and *injective* f
and *injective* h
shows *forest-components* $f =$ (*forest-components* $h * (d \sqcup d^T))^\star *$ *forest-components* h
⟨*proof*⟩

lemma *H-below-F*:
assumes $f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$

and *injective f*
and *injective h*
shows *forest-components h* ≤ *forest-components f*
⟨*proof*⟩

lemma *H-below-regular-g*:
assumes $f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$
and $f \leq --g$
and *symmetric g*
shows $h \leq --g$
⟨*proof*⟩

lemma *expression-equivalent-without-e-complement*:
assumes *selected-edge h j g* ≤ − *forest-components f*
shows $f \sqcap - (\mathit{selected\text{-}edge}\ h\ j\ g)^T \sqcap - (\mathit{path}\ f\ h\ j\ g) \sqcup (f \sqcap - (\mathit{selected\text{-}edge}\ h\ j\ g)^T \sqcap (\mathit{path}\ f\ h\ j\ g))^T \sqcup (\mathit{selected\text{-}edge}\ h\ j\ g)$
$= f \sqcap - (\mathit{path}\ f\ h\ j\ g) \sqcup (f \sqcap (\mathit{path}\ f\ h\ j\ g))^T \sqcup (\mathit{selected\text{-}edge}\ h\ j\ g)$
⟨*proof*⟩

The source of the *selected-edge* is contained in j, the vector describing those vertices still to be processed in the inner loop of Borůvka's algorithm.

lemma *et-below-j*:
assumes *vector j*
and *regular j*
and $j \neq bot$
shows *selected-edge h j g* ∗ *top* ≤ *j*
⟨*proof*⟩

4.4.1 Components of forests and forests modulo an equivalence

We prove a number of properties about *forest-modulo-equivalence* and *forest-components*.

lemma *component-single-eq*:
assumes *equivalence x*
shows *choose-component x v* = *x* ∗ *choose-component x v*
⟨*proof*⟩

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*⟩

There is a path in the *forest-modulo-equivalence* between edges a and b if and only if there is either a path in the *forest-modulo-equivalence* from a to b or one from a to c and one from c to b.

lemma *forest-modulo-equivalence-path-split-disj*:

assumes *equivalence H*
and *arc c*
and *regular a* ∧ *regular b* ∧ *regular c* ∧ *regular d* ∧ *regular H*
shows *forest-modulo-equivalence-path a b H* (*d* ⊔ *c*) ⟷
forest-modulo-equivalence-path a b H d ∨ (*forest-modulo-equivalence-path a c H d*
∧ *forest-modulo-equivalence-path c b H d*)
⟨*proof*⟩

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*⟩

lemma *forest-modulo-equivalence-d-U-e*:
assumes *forest f*
and *vector j*
and *regular j*
and *forest h*
and *forest-modulo-equivalence* (*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 *j* ≠ *bot*
shows *forest-modulo-equivalence* (*forest-components h*) (*d* ⊔ *selected-edge h j g*)
⟨*proof*⟩

4.4.2 Identifying arcs

The expression $d \sqcap \top e^\top H \sqcap (Hd^\top)^* Ha^\top \top$ identifies the edge incoming to the component that the *selected-edge*, *e*, is outgoing from and which is on the path from edge *a* to *e*. Here, we prove this expression is an *arc*.

lemma *shows-arc-x*:
assumes *forest-modulo-equivalence H d*
and *forest-modulo-equivalence-path 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*⟩

To maintain that *f* can be extended to a minimum spanning forest we

identify an edge, $i = v \sqcap \overline{F} e \top \sqcap \top e^\top F$, that may be exchanged with the *selected-edge*, e. Here, we show that i is an *arc*.

lemma *boruvka-edge-arc*:
  assumes *equivalence F*
    and *forest v*
    and *arc e*
    and *regular F*
    and $F \leq$ *forest-components* $(F \sqcap v)$
    and *regular v*
    and $v * e^T = bot$
    and $e * F * e = bot$
    and $e^T \leq v^\star$
    and $e \neq bot$
  shows *arc* $(v \sqcap -F * e * top \sqcap top * e^T * F)$
⟨*proof*⟩

4.4.3 Comparison of edge weights

In this section we compare the weight of the *selected-edge* with other edges of interest. For example, Theorem *e-leq-c-c-complement-transpose-general* is used to show that the *selected-edge* has its source inside and its target outside the component it is chosen for.

lemma *e-leq-c-c-complement-transpose-general*:
  assumes $e =$ *minarc* $(v * -(v)^T \sqcap g)$
    and *regular v*
  shows $e \leq v * -(v)^T$
⟨*proof*⟩

lemma *x-leq-c-transpose-general*:
  assumes *vector-classes x v*
    and $a^T * top \leq x * e * top$
    and $e \leq v * -v^T$
  shows $a \leq v^T$
⟨*proof*⟩

lemma *x-leq-c-complement-general*:
  assumes *vector v*
    and $v * v^T \leq x$
    and $a \leq v^T$
    and $a \leq -x$
  shows $a \leq -v$
⟨*proof*⟩

lemma *sum-e-below-sum-a-when-outgoing-same-component-general*:
  assumes $e =$ *minarc* $(v * -(v)^T \sqcap g)$
    and *symmetric g*
    and *arc a*
    and $a \leq -x \sqcap -- g$

and $a^T * top \leq x * e * top$
and *unique-vector-class x v*
shows *sum* $(e \sqcap g) \leq$ *sum* $(a \sqcap g)$
⟨*proof*⟩

lemma *sum-e-below-sum-x-when-outgoing-same-component*:
assumes *symmetric g*
and *vector j*
and *forest h*
and *regular h*
and $x \leq -$ *forest-components h* $\sqcap --$ *g*
and $x^T *$ *top* $\leq$ *forest-components h* $*$ *selected-edge h j g* $*$ *top*
and $j \neq$ *bot*
and *arc x*
shows *sum* (*selected-edge h j g* $\sqcap$ *g*) $\leq$ *sum* $(x \sqcap g)$
⟨*proof*⟩

If there is a path in the *forest-modulo-equivalence* from an edge between components, a, to the *selected-edge*, e, then the weight of e is no greater than the weight of a. This is because either,

- the edges a and e are adjacent the same component so that we can use *sum-e-below-sum-x-when-outgoing-same-component*, or
- there is at least one edge between a and e, namely x, the edge incoming to the component that e is outgoing from. The path from a to e is split on x using *forest-modulo-equivalence-path-split-disj*. We show that the weight of e is no greater than the weight of x by making use of lemma *sum-e-below-sum-x-when-outgoing-same-component*. We define x in a way that we can show that the weight of x is no greater than the weight of a using the invariant. Then, it follows that the weight of e is no greater than the weight of a owing to transitivity.

lemma *a-to-e-in-forest-modulo-equivalence*:
assumes *symmetric g*
and $f \leq --g$
and *vector j*
and *forest h*
and *forest-modulo-equivalence* (*forest-components h*) *d*
and $f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$
and $(\forall\ a\ b\ .$ *forest-modulo-equivalence-path a b* (*forest-components h*) $d \wedge a \leq -$(*forest-components h*) $\sqcap --\ g \wedge b \leq d \longrightarrow sum(b \sqcap g) \leq sum(a \sqcap g))$
and *regular d*
and $j \neq$ *bot*
and $b =$ *selected-edge h j g*
and *arc a*
and *forest-modulo-equivalence-path a b* (*forest-components h*) $(d \sqcup$ *selected-edge h j g*)
and $a \leq -$ *forest-components h* $\sqcap --$ *g*

and *regular h*
shows *sum* $(b \sqcap g) \leq$ *sum* $(a \sqcap g)$
⟨*proof*⟩

4.4.4 Maintenance of algorithm invariants

In this section, most of the work is done to maintain the invariants of the inner and outer loops of the algorithm. In particular, we use *exists-a-w* to maintain that f can be extended to a minimum spanning forest.

lemma *boruvka-exchange-spanning-inv*:
assumes *forest v*
and $v^\star * e^T = e^T$
and $i \leq v \sqcap top * e^T * w^{T\star}$
and *arc i*
and *arc e*
and $v \leq --g$
and $w \leq --g$
and $e \leq --g$
and *components g* $\leq$ *forest-components v*
shows $i \leq (v \sqcap -i)^{T\star} * e^T * top$
⟨*proof*⟩

lemma *exists-a-w*:
assumes *symmetric g*
and *forest f*
and $f \leq --g$
and *regular f*
and $(\exists w\ .$ *minimum-spanning-forest w g* $\wedge\ f \leq w \sqcup w^T)$
and *vector j*
and *regular j*
and *forest h*
and *forest-modulo-equivalence* (*forest-components h*) *d*
and $d * top \leq - j$
and *forest-components h* $* j = j$
and $f \sqcup f^T = h \sqcup h^T \sqcup d \sqcup d^T$
and $(\forall\ a\ b\ .$ *forest-modulo-equivalence-path a b* (*forest-components h*) *d* $\wedge\ a \leq -$(*forest-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-edge h j g*)$^T \sqcap -$ (*path f h j g*) $\sqcup$ ($f \sqcap -$ (*selected-edge h j g*)^T $\sqcap$ (*path f h j g*))$^T \sqcup$ (*selected-edge h j g*) $\leq w \sqcup w^T$
⟨*proof*⟩

lemma *boruvka-outer-invariant-when-e-not-bot*:

assumes *boruvka-inner-invariant j f h g d*
and $j \neq bot$
and *selected-edge h j g* $\leq -$ *forest-components f*
and *selected-edge h j g* $\neq bot$
shows *boruvka-outer-invariant* $(f \sqcap -$ *selected-edge h j* $g^T \sqcap -$ *path f h j g* $\sqcup (f \sqcap -$ *selected-edge h j* $g^T \sqcap$ *path f h j g*$)^T \sqcup$ *selected-edge h j g*$)$ *g*
⟨*proof*⟩

lemma *second-inner-invariant-when-e-not-bot*:
assumes *boruvka-inner-invariant j f h g d*
and $j \neq bot$
and *selected-edge h j g* $\leq -$ *forest-components f*
and *selected-edge h j g* $\neq bot$
shows *boruvka-inner-invariant*
$(j \sqcap -$ *choose-component* (*forest-components h*) *j*$)$
$(f \sqcap -$ *selected-edge h j* $g^T \sqcap -$ *path f h j g* $\sqcup$
$(f \sqcap -$ *selected-edge h j* $g^T \sqcap$ *path f h j g*$)^T \sqcup$
selected-edge h j g$)$
h g $(d \sqcup$ *selected-edge h j g*$)$
⟨*proof*⟩

lemma *second-inner-invariant-when-e-bot*:
assumes *selected-edge h j g* $= bot$
and *selected-edge h j g* $\leq -$ *forest-components f*
and *boruvka-inner-invariant j f h g d*
shows *boruvka-inner-invariant*
$(j \sqcap -$ *choose-component* (*forest-components h*) *j*$)$
$(f \sqcap -$ *selected-edge h j* $g^T \sqcap -$ *path f h j g* $\sqcup$
$(f \sqcap -$ *selected-edge h j* $g^T \sqcap$ *path f h j g*$)^T \sqcup$
selected-edge h j g$)$
h g $(d \sqcup$ *selected-edge h j g*$)$
⟨*proof*⟩

4.5 Formalization and correctness proof

The following result shows that Borůvka's algorithm constructs a minimum spanning forest. We have the same postcondition as the 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*) $\sqcap$ *g* $\neq$ *bot*
INV { *boruvka-outer-invariant f g* }
DO
j := *top*;
h := *f*;
d := *bot*;
WHILE j $\neq$ *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⟩

end

end
```

References

[1] O. Borůvka. O jistém problému minimálním. *Práce moravské přírodovědecké společnosti*, 3(3):37–58, 1926.

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

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

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

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

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

[7] J. Nešetřil, E. Milková, and H. Nešetřilová. Otakar Borůvka on minimum spanning tree problem – Translation of both the 1926 papers, comments, history. *Discrete Mathematics*, 233(1–3):3–36, 2001.

[8] N. Robinson-O'Brien. A formal correctness proof of Borůvka's minimum spanning tree algorithm. Master's thesis, University of Canterbury, 2020. https://doi.org/10.26021/10196.