An Axiomatic Characterization of the Single-Source Shortest Path Problem

By Christine Rizkallah

December 14, 2021

Abstract

This theory is split into two sections. In the first section, we give a formal proof that a well-known axiomatic characterization of the single-source shortest path problem is correct. Namely, we prove that in a directed graph $G = (V, E)$ with a non-negative cost function on the edges the single-source shortest path function $\mu : V \to \mathbb{R} \cup \{\infty\}$ is the only function that satisfies a set of four axioms. The first axiom states that the distance from the source vertex s to itself should be equal to zero. The second states that the distance from s to a vertex $v \in V$ should be infinity if and only if there is no path from s to v. The third axiom is called triangle inequality and states that if there is a path from s to v, and an edge $(u, v) \in E$, the distance from s to v is less than or equal to the distance from s to u plus the cost of (u, v). The last axiom is called justification, it states that for every vertex v other than s, if there is a path p from s to v in G, then there is a predecessor edge (u, v) on p such that the distance from s to v is equal to the distance from s to u plus the cost of (u, v).

In the second section, we give a formal proof of the correctness of an axiomatic characterization of the single-source shortest path problem for directed graphs with general cost functions $c : E \to \mathbb{R}$. The axioms here are more involved because we have to account for potential negative cycles in the graph. The axioms are summarized in the three isabelle locales.

Contents

theory *ShortestPath*
imports
Graph-Theory.Graph-Theory
begin

1 Shortest Path (with non-negative edge costs)

The following theory is used in the verification of a certifying algorithm's checker for shortest path. For more information see [1].

locale *basic-sp* =
  fin-digraph +
  fixes *dist* :: *'a* ⇒ *ereal*
  fixes *c* :: *'b* ⇒ *real*
  fixes *s* :: *'a*
  assumes *general-source-val*: *dist s* ≤ *0*
  assumes *trian*:
    ⋀*e*. *e* ∈ *arcs G* ⟹
      dist (*head G e*) ≤ *dist* (*tail G e*) + *c e*

locale *basic-just-sp* =
  basic-sp +
  fixes *num* :: *'a* ⇒ *enat*
  assumes *just*:
    ⋀*v*. ⟦*v* ∈ *verts G*; *v* ≠ *s*; *num v* ≠ ∞⟧ ⟹
      ∃ *e* ∈ *arcs G*. *v* = *head G e* ∧
        dist v = *dist* (*tail G e*) + *c e* ∧
        num v = *num* (*tail G e*) + (*enat 1*)

locale *shortest-path-pos-cost* =
  basic-just-sp +
  assumes *s-in-G*: *s* ∈ *verts G*
  assumes *tail-val*: *dist s* = *0*
  assumes *no-path*: ⋀*v*. *v* ∈ *verts G* ⟹ *dist v* = ∞ ⟷ *num v* = ∞
  assumes *pos-cost*: ⋀*e*. *e* ∈ *arcs G* ⟹ *0* ≤ *c e*

locale *basic-just-sp-pred* =
  basic-sp +
  fixes *num* :: *'a* ⇒ *enat*
  fixes *pred* :: *'a* ⇒ *'b option*
  assumes *just*:
    ⋀*v*. ⟦*v* ∈ *verts G*; *v* ≠ *s*; *num v* ≠ ∞⟧ ⟹
      ∃ *e* ∈ *arcs G*.
        e = *the* (*pred v*) ∧
        v = *head G e* ∧
        dist v = *dist* (*tail G e*) + *c e* ∧
        num v = *num* (*tail G e*) + (*enat 1*)

sublocale *basic-just-sp-pred* ⊆ *basic-just-sp*
using *basic-just-sp-pred-axioms*
unfolding *basic-just-sp-pred-def*
  basic-just-sp-pred-axioms-def
by *unfold-locales* (*blast*)

locale *shortest-path-pos-cost-pred* =

basic-just-sp-pred +
assumes *s-in-G*: $s \in verts\ G$
assumes *tail-val*: $dist\ s = 0$
assumes *no-path*: $\bigwedge v.\ v \in verts\ G \Longrightarrow dist\ v = \infty \longleftrightarrow num\ v = \infty$
assumes *pos-cost*: $\bigwedge e.\ e \in arcs\ G \Longrightarrow 0 \leq c\ e$

sublocale *shortest-path-pos-cost-pred* $\subseteq$ *shortest-path-pos-cost*
using *shortest-path-pos-cost-pred-axioms*
by *unfold-locales*
(*auto simp*: *shortest-path-pos-cost-pred-def*
shortest-path-pos-cost-pred-axioms-def)

lemma *tail-value-helper*:
assumes $hd\ p = last\ p$
assumes *distinct p*
assumes $p \neq []$
shows $p = [hd\ p]$
by (*metis assms distinct.simps(2) list.sel(1) neq-Nil-conv last-ConsR last-in-set*)

lemma (**in** *basic-sp*) *dist-le-cost*:
fixes $v :: 'a$
fixes $p :: 'b\ list$
assumes *awalk s p v*
shows $dist\ v \leq awalk\text{-}cost\ c\ p$
using *assms*
proof (*induct length p arbitrary*: *p v*)
case *0*
hence $s = v$ **by** *auto*
thus *?case* **using** *0(1) general-source-val*
by (*metis awalk-cost-Nil length-0-conv zero-ereal-def*)
next
case (*Suc n*)
then obtain $p'\ e$ **where** $p'e$: $p = p' @ [e]$
by (*cases p rule*: *rev-cases*) *auto*
then obtain *u* **where** *ewu*: $awalk\ s\ p'\ u \wedge awalk\ u\ [e]\ v$
using *awalk-append-iff Suc(3)* **by** *simp*
then have *du*: $dist\ u \leq ereal\ (awalk\text{-}cost\ c\ p')$
using *Suc p'e* **by** *simp*
from *ewu* **have** *ust*: $u = tail\ G\ e$ **and** *vta*: $v = head\ G\ e$
by *auto*
then have $dist\ v \leq dist\ u + c\ e$
using *ewu du ust trian*[**where** *e*=*e*] **by** *force*
with *du* **have** $dist\ v \leq ereal\ (awalk\text{-}cost\ c\ p') + c\ e$
by (*metis add-right-mono order-trans*)
thus $dist\ v \leq awalk\text{-}cost\ c\ p$
using *awalk-cost-append p'e* **by** *simp*
qed

lemma (**in** *fin-digraph*) *witness-path*:

```
  assumes μ c s v = ereal r
  shows ∃ p. apath s p v ∧ μ c s v = awalk-cost c p
proof −
  have sv: s →* v
    using shortest-path-inf[of s v c] assms by fastforce
  {
    fix p assume awalk s p v
    then have no-neg-cyc:
    ¬ (∃ w q. awalk w q w ∧ w ∈ set (awalk-verts s p) ∧ awalk-cost c q < 0)
      using neg-cycle-imp-inf-μ assms by force
  }
  thus ?thesis using no-neg-cyc-reach-imp-path[OF sv] by presburger
qed

lemma (in basic-sp) dist-le-μ:
  fixes v :: 'a
  assumes v ∈ verts G
  shows dist v ≤ μ c s v
proof (rule ccontr)
  assume nt: ¬ ?thesis
  show False
  proof (cases μ c s v)
    show ⋀r. μ c s v = ereal r ⟹ False
    proof −
      fix r assume r-asm: μ c s v = ereal r
      hence sv: s →* v
        using shortest-path-inf[where u=s and v=v and f=c] by auto
      obtain p where
        awalk s p v
        μ c s v = awalk-cost c p
        using witness-path[OF r-asm] unfolding apath-def by force
      thus False using nt dist-le-cost by simp
    qed
  next
    show μ c s v = ∞ ⟹ False using nt by simp
  next
    show μ c s v = − ∞ ⟹ False using dist-le-cost
    proof −
      assume asm: μ c s v = − ∞
      let ?C = (λx. ereal (awalk-cost c x)) ` {p. awalk s p v}
      have ∃ x∈ ?C. x < dist v
        using nt unfolding μ-def not-le INF-less-iff by simp
      then obtain p where
        awalk s p v
        awalk-cost c p < dist v
        by force
      thus False using dist-le-cost by force
    qed
  qed
```

qed

lemma (**in** *basic-just-sp*) *dist-ge-μ*:
fixes $v :: 'a$
assumes $v \in verts\ G$
assumes $num\ v \neq \infty$
assumes $dist\ v \neq -\infty$
assumes $\mu\ c\ s\ s = ereal\ 0$
assumes $dist\ s = 0$
assumes $\bigwedge u.\ u \in verts\ G \Longrightarrow u \neq s \Longrightarrow$
$num\ u \neq \infty \Longrightarrow num\ u \neq enat\ 0$
shows $dist\ v \geq \mu\ c\ s\ v$
proof −
obtain n **where** $enat\ n = num\ v$ **using** *assms*(*2*) **by** *force*
thus *?thesis* **using** *assms*
proof(*induct n arbitrary*: *v*)
case *0* **thus** *?case* **by** (*cases v=s*, *auto*)
next
case (*Suc n*)
thus *?case*
proof (*cases v=s*)
case *False*
obtain *e* **where** *e-assms*:
$e \in arcs\ G$
$v = head\ G\ e$
$dist\ v = dist\ (tail\ G\ e) + ereal\ (c\ e)$
$num\ v = num\ (tail\ G\ e) + enat\ 1$
using *just*[*OF Suc*(*3*) *False Suc*(*4*)] **by** *blast*
then have *nsinf*:$num\ (tail\ G\ e) \neq \infty$
by (*metis Suc*(*2*) *enat.simps*(*3*) *enat-1 plus-enat-simps*(*2*))
then have *ns*:$enat\ n = num\ (tail\ G\ e)$
using *e-assms*(*4*) *Suc*(*2*) **by** *force*
have *ds*: $dist\ (tail\ G\ e) = \mu\ c\ s\ (tail\ G\ e)$
using *Suc*(*1*)[*OF ns tail-in-verts*[*OF e-assms*(*1*)] *nsinf*]
Suc(*5*−*8*) *e-assms*(*3*) *dist-le-μ*[*OF tail-in-verts*[*OF e-assms*(*1*)]]
by *simp*
have *dmuc*:$dist\ v \geq \mu\ c\ s\ (tail\ G\ e) + ereal\ (c\ e)$
using *e-assms*(*3*) *ds* **by** *auto*
thus *?thesis*
proof (*cases dist v* = ∞)
case *False*
have *arc-to-ends G e* = (*tail G e*, *v*)
unfolding *arc-to-ends-def*
by (*simp add*: *e-assms*(*2*))
obtain *r* **where** μr: $\mu\ c\ s\ (tail\ G\ e) = ereal\ r$
using *e-assms*(*3*) *Suc*(*5*) *ds False*
by (*cases μ c s* (*tail G e*), *auto*)
obtain *p* **where**
awalk s p (*tail G e*) **and**

```
      μs: μ c s (tail G e) = ereal (awalk-cost c p)
      using witness-path[OF μr] unfolding apath-def
      by blast
    then have pe: awalk s (p @ [e]) v
      using e-assms(1,2) by (auto simp: awalk-simps)
    hence muc:μ c s v ≤ μ c s (tail G e) + ereal (c e)
    using μs min-cost-le-walk-cost[OF pe] by simp
    thus  dist v ≥ μ c s v  using dmuc by simp
  qed simp
 qed (simp add: Suc(6,7))
 qed
qed

lemma (in shortest-path-pos-cost) tail-value-check:
 fixes u :: 'a
 assumes s ∈ verts G
 shows μ c s s = ereal 0
proof −
 have ∗: awalk s [] s using assms unfolding awalk-def by simp
 hence μ c s s ≤ ereal 0 using min-cost-le-walk-cost[OF ∗] by simp
 moreover
 have (⋀p. awalk s p s ⟹ ereal(awalk-cost c p) ≥ ereal 0)
  using pos-cost pos-cost-pos-awalk-cost by auto
 hence μ c s s ≥ ereal 0
  unfolding μ-def by (blast intro: INF-greatest)
 ultimately
 show ?thesis by simp
qed

lemma (in shortest-path-pos-cost) num-not0:
 fixes v :: 'a
 assumes v ∈ verts G
 assumes v ≠ s
 assumes num v ≠ ∞
 shows num v ≠ enat 0
proof −
 obtain ku where num v = ku + enat 1
  using assms just by blast
 thus ?thesis  by (induct ku) auto
qed

lemma (in shortest-path-pos-cost) dist-ne-ninf:
 fixes v :: 'a
 assumes v ∈ verts G
 shows dist v ≠ −∞
proof (cases num v = ∞)
case False
 obtain n where enat n = num v
  using False by force
```

```
  thus ?thesis using assms False
  proof(induct n arbitrary: v)
  case 0 thus ?case
    using num-not0 tail-val by (cases v=s, auto)
  next
  case (Suc n)
    thus ?case
    proof (cases v=s)
    case True
      thus ?thesis using tail-val by simp
    next
    case False
      obtain e where e-assms:
        e ∈ arcs G
        dist v = dist (tail G e) + ereal (c e)
        num v = num (tail G e) + enat 1
        using just[OF Suc(3) False Suc(4)] by blast
      then have nsinf:num (tail G e) ≠ ∞
        by (metis Suc(2) enat.simps(3) enat-1 plus-enat-simps(2))
      then have ns:enat n = num (tail G e)
        using e-assms(3) Suc(2) by force
      have dist (tail G e ) ≠ − ∞
        by (rule Suc(1) [OF ns tail-in-verts[OF e-assms(1)] nsinf])
      thus ?thesis using e-assms(2) by simp
    qed
  qed
next
case True
  thus ?thesis using no-path[OF assms] by simp
qed

theorem (in shortest-path-pos-cost) correct-shortest-path:
  fixes v :: 'a
  assumes v ∈ verts G
  shows dist v = μ c s v
  using no-path[OF assms(1)] dist-le-μ[OF assms(1)]
    dist-ge-μ[OF assms(1) - dist-ne-ninf[OF assms(1)]
    tail-value-check[OF s-in-G] tail-val num-not0]
    by fastforce

corollary (in shortest-path-pos-cost-pred) correct-shortest-path-pred:
  fixes v :: 'a
  assumes v ∈ verts G
  shows dist v = μ c s v
  using correct-shortest-path assms by simp

end
theory ShortestPathNeg
```

imports *ShortestPath*

begin

2 Shortest Path (with general edge costs)

locale *shortest-paths-locale-step1* =
  fixes G :: $('a, 'b)$ *pre-digraph* (**structure**)
  fixes s :: $'a$
  fixes c :: $'b \Rightarrow$ *real*
  fixes *num* :: $'a \Rightarrow$ *nat*
  fixes *parent-edge* :: $'a \Rightarrow$ $'b$ *option*
  fixes *dist* :: $'a \Rightarrow$ *ereal*
  assumes *graphG*: *fin-digraph G*
  assumes *s-assms*:
    $s \in$ *verts G*
    dist $s \neq \infty$
    parent-edge s = *None*
    num s = *0*
  assumes *parent-num-assms*:
    $\bigwedge v.\ [\![v \in$ *verts G*; $v \neq s$; *dist* $v \neq \infty]\!] \Longrightarrow$
    $(\exists\, e \in$ *arcs G*. *parent-edge* v = *Some* $e \wedge$
    head G $e = v \wedge$ *dist* (*tail G* e) $\neq \infty \wedge$
    num v = *num* (*tail G* e) + *1*)
  assumes *noPedge*: $\bigwedge e.\ e \in$ *arcs G* $\Longrightarrow$
    dist (*tail G* e) $\neq \infty \Longrightarrow$ *dist* (*head G* e) $\neq \infty$

sublocale *shortest-paths-locale-step1* $\subseteq$ *fin-digraph G*
  using *graphG* **by** *auto*

definition (**in** *shortest-paths-locale-step1*) *enum* :: $'a \Rightarrow$ *enat* **where**
  enum v = (*if* (*dist* $v = \infty \vee$ *dist* $v = -\infty$) *then* ∞ *else num* v)

locale *shortest-paths-locale-step2* =
  shortest-paths-locale-step1 +
  basic-just-sp G dist c s enum +
  assumes *source-val*: $(\exists\, v \in$ *verts G*. *enum* $v \neq \infty) \Longrightarrow$ *dist* s = *0*
  assumes *no-edge-Vm-Vf*:
    $\bigwedge e.\ e \in$ *arcs G* $\Longrightarrow$ *dist* (*tail G* e) $= -\infty \Longrightarrow \forall\ r.$ *dist* (*head G* e) $\neq$ *ereal* r

function (**in** *shortest-paths-locale-step1*) *pwalk* :: $'a \Rightarrow$ $'b$ *list*
where
  pwalk v =
    (*if* ($v = s \vee$ *dist* $v = \infty \vee v \notin$ *verts G*)
      then []
      else pwalk (*tail G* (*the* (*parent-edge* v))) @ [*the* (*parent-edge* v)]
)
by *auto*

termination (**in** *shortest-paths-locale-step1*)
using *parent-num-assms*
by (*relation measure num*, *auto*, *fastforce*)

lemma (**in** *shortest-paths-locale-step1*) *pwalk-simps*:
$v = s \vee dist\ v = \infty \vee v \notin verts\ G \Longrightarrow pwalk\ v = []$
$v \neq s \Longrightarrow dist\ v \neq \infty \Longrightarrow v \in verts\ G \Longrightarrow$
$pwalk\ v = pwalk\ (tail\ G\ (the\ (parent\text{-}edge\ v))) @ [the\ (parent\text{-}edge\ v)]$
by *auto*

definition (**in** *shortest-paths-locale-step1*) *pwalk-verts* :: $'a \Rightarrow 'a\ set$ **where**
$pwalk\text{-}verts\ v = \{u.\ u \in set\ (awalk\text{-}verts\ s\ (pwalk\ v))\}$

locale *shortest-paths-locale-step3* =
shortest-paths-locale-step2 +
fixes $C :: ('a \times ('b\ awalk))\ set$
assumes *C-se*:
$C \subseteq \{(u, p).\ dist\ u \neq \infty \wedge awalk\ u\ p\ u \wedge awalk\text{-}cost\ c\ p < 0\}$
assumes *int-neg-cyc*:
$\bigwedge v.\ v \in verts\ G \Longrightarrow dist\ v = -\infty \Longrightarrow$
$(fst\ `\ C) \cap pwalk\text{-}verts\ v \neq \{\}$

locale *shortest-paths-locale-step2-pred* =
shortest-paths-locale-step1 +
fixes *pred* :: $'a \Rightarrow 'b\ option$
assumes *bj*: *basic-just-sp-pred G dist c s enum pred*
assumes *source-val*: $(\exists v \in verts\ G.\ enum\ v \neq \infty) \Longrightarrow dist\ s = 0$
assumes *no-edge-Vm-Vf*:
$\bigwedge e.\ e \in arcs\ G \Longrightarrow dist\ (tail\ G\ e) = -\infty \Longrightarrow \forall\ r.\ dist\ (head\ G\ e) \neq ereal\ r$

lemma (**in** *shortest-paths-locale-step1*) *num-s-is-min*:
assumes $v \in verts\ G$
assumes $v \neq s$
assumes $dist\ v \neq \infty$
shows $num\ v > 0$
using *parent-num-assms*[*OF assms*] **by** *fastforce*

lemma (**in** *shortest-paths-locale-step1*) *path-from-root-Vr-ex*:
fixes $v :: 'a$
assumes $v \in verts\ G$
assumes $v \neq s$
assumes $dist\ v \neq \infty$
shows $\exists e.\ s \rightarrow^* tail\ G\ e \wedge$
$e \in arcs\ G \wedge head\ G\ e = v \wedge dist\ (tail\ G\ e) \neq \infty \wedge$
$parent\text{-}edge\ v = Some\ e \wedge num\ v = num\ (tail\ G\ e) + 1$
using *assms*
proof(*induct num v − 1 arbitrary* : *v*)

case *0*
obtain *e* **where** *ee*:
e ∈ *arcs G head G e* = *v dist* (*tail G e*) ≠ ∞
parent-edge v = *Some e num v* = *num* (*tail G e*) + *1*
using *parent-num-assms*[*OF 0*(*2*−*4*)] **by** *fast*
have *tail G e* = *s*
using *num-s-is-min*[*OF tail-in-verts* [*OF ee*(*1*)] - *ee*(*3*)]
ee(*5*) *0*(*1*) **by** *auto*
then show *?case* **using** *ee* **by** *auto*
next
case (*Suc n*′)
obtain *e* **where** *ee*:
e ∈ *arcs G head G e* = *v dist* (*tail G e*) ≠ ∞
parent-edge v = *Some e num v* = *num* (*tail G e*) + *1*
using *parent-num-assms*[*OF Suc*(*3*−*5*)] **by** *fast*
then have *ss*: *tail G e* ≠ *s*
using *num-s-is-min tail-in-verts*
Suc(*2*) *s-assms*(*4*) **by** *force*
have *nst*: *n*′ = *num* (*tail G e*) − *1*
using *ee*(*5*) *Suc*(*2*) **by** *presburger*
obtain *e*′ **where** *reach*: *s* →* *tail G e*′ **and**
e′: *e*′ ∈ *arcs G head G e*′ = *tail G e dist* (*tail G e*′) ≠ ∞
using *Suc*(*1*)[*OF nst tail-in-verts*[*OF ee*(*1*)] *ss ee*(*3*)] **by** *blast*
then have *s* →* *tail G e*
by (*metis arc-implies-awalk reachable-awalk reachable-trans*)
then show *?case* **using** *e*′ *ee* **by** *auto*
qed

lemma (**in** *shortest-paths-locale-step1*) *path-from-root-Vr*:
fixes *v* :: *'a*
assumes *v* ∈ *verts G*
assumes *dist v* ≠ ∞
shows *s* →* *v*
proof(*cases v* = *s*)
case *True* **thus** *?thesis* **using** *assms* **by** *simp*
next
case *False*
obtain *e* **where** *s* →* *tail G e e* ∈ *arcs G head G e* = *v*
using *path-from-root-Vr-ex*[*OF assms*(*1*) *False assms*(*2*)] **by** *blast*
then have *s* →* *tail G e tail G e* → *v*
by (*auto intro*: *in-arcs-imp-in-arcs-ends*)
then show *?thesis* **by** (*rule reachable-adj-trans*)
qed

lemma (**in** *shortest-paths-locale-step1*) *μ-V-less-inf*:
fixes *v* :: *'a*
assumes *v* ∈ *verts G*
assumes *dist v* ≠ ∞
shows *μ c s v* ≠ ∞

```
  using assms path-from-root-Vr μ-reach-conv by force

lemma (in shortest-paths-locale-step2) enum-not0:
  assumes v ∈ verts G
  assumes v ≠ s
  assumes enum v ≠ ∞
  shows enum v ≠ enat 0
    using parent-num-assms[OF assms(1,2)] assms unfolding enum-def by auto

lemma (in shortest-paths-locale-step2) dist-Vf-μ:
  fixes v :: 'a
  assumes vG: v ∈ verts G
  assumes ∃ r. dist v = ereal r
  shows dist v = μ c s v
proof −
  have ds: dist s =  0
    using assms source-val unfolding enum-def by force
  have ews:awalk s [] s
    using s-assms(1) unfolding awalk-def by simp
  have mu: μ c s s = ereal 0
    using min-cost-le-walk-cost[OF ews, where c=c]
    awalk-cost-Nil ds  dist-le-μ[OF s-assms(1)] zero-ereal-def
    by simp
  thus ?thesis
    using ds assms dist-le-μ[OF vG]
    dist-ge-μ[OF vG - - mu ds enum-not0]
    unfolding enum-def by fastforce
qed

lemma (in shortest-paths-locale-step1) pwalk-awalk:
  fixes v :: 'a
  assumes v ∈ verts G
  assumes dist v ≠ ∞
  shows awalk s (pwalk v) v
proof (cases v=s)
case True
  thus ?thesis
    using assms pwalk.simps[where v=v]
    awalk-Nil-iff by presburger
next
case False
  from assms show ?thesis
  proof (induct rule: pwalk.induct)
    fix v
    let ?e = the (parent-edge v)
    let ?u = tail G ?e
    assume ewu: ¬ (v = s ∨ dist v = ∞ ∨ v ∉ verts G) ⟹
                ?u ∈ verts G ⟹ dist ?u ≠ ∞ ⟹
                awalk s (pwalk ?u) ?u
```

```
    assume vG: v ∈ verts G
    assume dv: dist v ≠ ∞
    thus awalk s (pwalk v) v
    proof (cases v = s ∨ dist v = ∞ ∨ v ∉ verts G)
    case True
      thus ?thesis
        using pwalk.simps vG dv
        awalk-Nil-iff by fastforce
    next
    case False
      obtain e  where ee:
        e ∈arcs G
        parent-edge v = Some e
        head G e = v
        dist (tail G e) ≠ ∞
        using parent-num-assms False by blast
      hence awalk s (pwalk ?u) ?u
        using ewu[OF False] tail-in-verts by simp
      hence awalk s (pwalk (tail G e) @ [e]) v
        using ee(1−3) vG
        by (auto simp: awalk-simps simp del: pwalk.simps)
      also have pwalk (tail G e) @ [e] = pwalk v
        using False ee(2) unfolding pwalk.simps[where v=v] by auto
      finally show ?thesis .
    qed
  qed
qed

lemma (in shortest-paths-locale-step3) μ-ninf:
  fixes v :: 'a
  assumes v ∈ verts G
  assumes dist v = − ∞
  shows μ c s v = − ∞
proof −
  have awalk s (pwalk v) v
    using pwalk-awalk assms by force
moreover
  obtain w where ww: w ∈ fst ‘ C ∩ pwalk-verts v
    using int-neg-cyc[OF assms] by blast
  then obtain q where
     awalk w q w
     awalk-cost c q < 0
     using C-se by auto
moreover
  have w ∈ set (awalk-verts s (pwalk v))
    using ww unfolding pwalk-verts-def by fast
ultimately
  show ?thesis using  neg-cycle-imp-inf-μ by force
qed
```

lemma (**in** *shortest-paths-locale-step3*) *correct-shortest-path*:
fixes $v :: 'a$
assumes $v \in verts\ G$
shows $dist\ v = \mu\ c\ s\ v$
proof(*cases dist v*)
show $\bigwedge r.\ dist\ v = ereal\ r \Longrightarrow dist\ v = \mu\ c\ s\ v$
using *dist-Vf-μ*[*OF assms*] **by** *simp*
next
show $dist\ v = \infty \Longrightarrow dist\ v = \mu\ c\ s\ v$
using *μ-V-less-inf*[*OF assms*]
dist-le-μ[*OF assms*] **by** *simp*
next
show $dist\ v = -\infty \Longrightarrow dist\ v = \mu\ c\ s\ v$
using *μ-ninf*[*OF assms*] **by** *simp*
qed

end

References

[1] E. Alkassar, S. Böhme, K. Mehlhorn, and C. Rizkallah. A framework for the verification of certifying computations. *Journal of Automated Reasoning*, 2013. To Appear.