# An Axiomatic Characterization of Shortest-Path in Isabelle/HOL

By Christine Rizkallah

January 4, 2013

**Abstract**

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 negative cycles in the graph. The axioms are summarized in the three isabelle locales at the beginning of the section.

## Contents



**theory** *ShortestPath*
**imports**
*../../graph-library/Graphs*

**begin**

# 1 Shortest Path (non-negative edge costs)

**locale** *basic-sp* =
  *pseudo-digraph* +
  **fixes** *dist* :: $'a \Rightarrow$ *ereal*
  **fixes** *c* :: $'b \Rightarrow$ *real*
  **fixes** *s* :: $'a$
  **assumes** *general-source-val*: $dist\ s \leq 0$
  **assumes** *trian*:
    $\bigwedge e.\ e \in edges\ G \Longrightarrow$
      $dist\ (target\ G\ e) \leq dist\ (start\ G\ e) + c\ e$

**locale** *basic-just-sp* =
  *basic-sp* +
  **fixes** *num* :: $'a \Rightarrow$ *enat*
  **assumes** *just*:
    $\bigwedge v.\ [\![v \in verts\ G;\ v \neq s;\ num\ v \neq \infty]\!] \Longrightarrow$
      $\exists\ e \in edges\ G.\ v = target\ G\ e\ \wedge$
        $dist\ v = dist\ (start\ G\ e) + c\ e\ \ \wedge$
        $num\ v = num\ (start\ G\ e) + (enat\ 1)$

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

**locale** *basic-just-sp-pred* =
  *basic-sp* +
  **fixes** *num* :: $'a \Rightarrow$ *enat*
  **fixes** *pred* :: $'a \Rightarrow 'b$ *option*
  **assumes** *just*:
    $\bigwedge v.\ [\![v \in verts\ G;\ v \neq s;\ num\ v \neq \infty]\!] \Longrightarrow$
      $\exists\ e \in edges\ G.$
        $e = the\ (pred\ v)\ \wedge$
        $v = target\ G\ e\ \wedge$
        $dist\ v = dist\ (start\ G\ e) + c\ e\ \ \wedge$
        $num\ v = num\ (start\ G\ e) + (enat\ 1)$

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

**sublocale** *shortest-path-pos-cost-pred* ⊆ *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** *start-value-helper*:
**assumes** $hd\ p = last\ p$
**assumes** *distinct p*
**assumes** $p \neq []$
**shows** $p = [hd\ p]$
**by** (*metis assms distinct.simps*(*2*) *hd.simps neq-Nil-conv last-ConsR last-in-set*)

**lemma** (**in** *basic-sp*) *dist-le-cost*:
**fixes** $v :: 'a$
**fixes** $p :: 'b\ list$
**assumes** *ewalk s p v*
**shows** $dist\ v \leq ewalk\text{-}cost\ c\ p$
**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 ewalk-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*: $ewalk\ s\ p'\ u \wedge ewalk\ u\ [e]\ v$
**using** *ewalk-join-decomp Suc*(*3*) **by** *simp*
**then have** *du*: $dist\ u \leq ereal\ (ewalk\text{-}cost\ c\ p')$
**using** *Suc* $p'e$ **by** *simp*
**from** *ewu* **have** *ust*: $u = start\ G\ e$ **and** *vta*: $v = target\ G\ e$
**by** *auto*
**then have** $dist\ v \leq dist\ u + c\ e$
**using** *ewu du ust trian*[**where** *e*=*e*] **by** *force*
**with** *du* **have** $dist\ v \leq ereal\ (ewalk\text{-}cost\ c\ p') + c\ e$
**by** (*metis add-right-mono order-trans*)
**thus** $dist\ v \leq ewalk\text{-}cost\ c\ p$
**using** *ewalk-cost-append* $p'e$ **by** *simp*
**qed**

**lemma** (**in** *pseudo-digraph*) *witness-path*:
**assumes** $\mu\ c\ s\ v = ereal\ r$
**shows** $\exists\ p.\ epath\ s\ p\ v \wedge \mu\ c\ s\ v = ewalk\text{-}cost\ c\ p$

```
proof -
  have sv: s →∗G v
    using shortest-path-inf assms by fastforce
  {
    fix p assume ewalk s p v
    then have no-neg-cyc:
    ¬ (∃ w q. ewalk w q w ∧ w ∈ set (ewalk-verts s p) ∧ ewalk-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 →∗G v
        using shortest-path-inf[where u=s and v=v and f=c] by auto
      obtain p where
        ewalk s p v
        μ c s v = ewalk-cost c p
        using witness-path[OF r-asm] unfolding epath-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 (ewalk-cost c x)) ` {p. ewalk s p v}
      have ∃ x∈ ?C. x < dist v
        using Inf-ereal-iff [where y =dist vand X=?C and z= −∞]
        nt asm unfolding μ-def INF-def by simp
      then obtain p where
        ewalk s p v
        ewalk-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 ∈ verts G
  assumes num v ≠ ∞
  assumes dist v ≠ −∞
  assumes μ c s s = ereal 0
  assumes dist s = 0
  assumes ⋀u. u∈verts G ⟹ u≠s ⟹
          num u ≠ ∞ ⟹ num u ≠ enat 0
  shows dist v ≥ μ 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 ∈ edges G
        v = target G e
        dist v = dist (start G e) + ereal (c e)
        num v = num (start G e) + enat 1
        using just[OF Suc(3) False Suc(4)] by blast
      then have nsinf:num (start G e) ≠ ∞
        by (metis Suc(2) enat.simps(3) enat-1 plus-enat-simps(2))
      then have ns:enat n = num (start G e)
        using e-assms(4) Suc(2) by force
      have  ds: dist (start G e) = μ c s (start G e)
        using Suc(1)[OF ns start-in-verts[OF e-assms(1)] nsinf]
        Suc(5−8) e-assms(3) dist-le-μ[OF start-in-verts[OF e-assms(1)]]
        by simp
      have dmuc:dist v ≥ μ c s (start G e) + ereal (c e)
        using e-assms(3) ds  by auto
      thus ?thesis
      proof (cases dist v = ∞)
      case False
        have edge-to-ends G e = (start G e, v)
          unfolding edge-to-ends-def
          by (simp add: e-assms(2))
        obtain r where  μr: μ c s (start G e) = ereal r
           using e-assms(3) Suc(5) ds False
           by (cases μ c s (start G e), auto)
        obtain p where
          ewalk s p (start G e) and
          μs: μ c s (start G e) = ereal (ewalk-cost c p)
```

**using** *witness-path*[*OF* $\mu r$] **unfolding** *epath-def*
**by** *blast*
**then have** *pe*: *ewalk s* (*p* @ [*e*]) *v*
**using** *e-assms*(*1*,*2*) *ewalk-Cons-iff ewalk-empty-iff*
*ewalk-ewalk-joinI Suc*(*3*) **by** (*cases p* @ [*e*], *blast+*)
**hence** *muc*:$\mu$ *c s v* $\leq$ $\mu$ *c s* (*start G e*) + *ereal* (*c e*)
**using** $\mu s$ *min-cost-le-walk-cost*[*OF pe*] **by** *simp*
**thus** *dist v* $\geq$ $\mu$ *c s v* **using** *dmuc* **by** *simp*
**qed** *simp*
**qed** (*simp add*: *Suc*(*6*,*7*))
**qed**
**qed**

**lemma** (**in** *shortest-path-pos-cost*) *start-value-check*:
**fixes** *u* :: $'a$
**assumes** *s* $\in$ *verts G*
**shows** $\mu$ *c s s* = *ereal 0*
**proof** −
**have** $*$: *ewalk s* [] *s* **using** *assms* **unfolding** *ewalk-def* **by** *simp*
**hence** $\mu$ *c s s* $\leq$ *ereal 0* **using** *min-cost-le-walk-cost*[*OF* $*$] **by** *simp*
**moreover**
**have** ($\bigwedge p$. *ewalk s p s* $\Longrightarrow$ *ereal*(*ewalk-cost c p*) $\geq$ *ereal 0*)
**using** *pos-cost pos-cost-pos-ewalk-cost* **by** *auto*
**hence** $\mu$ *c s s* $\geq$ *ereal 0*
**unfolding** $\mu$*-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* $\in$ *verts G*
**assumes** *v* $\neq$ *s*
**assumes** *num v* $\neq$ $\infty$
**shows** *num v* $\neq$ *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* $\in$ *verts G*
**shows** *dist v* $\neq$ $-\infty$
**proof** (*cases num v* = $\infty$)
**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 start-val by (cases v=s, auto)
  next
  case (Suc n)
    thus ?case
    proof (cases v=s)
    case True
      thus ?thesis using start-val by simp
    next
    case False
      obtain e where e-assms:
        e ∈ edges G
        dist v = dist (start G e) + ereal (c e)
        num v = num (start G e) + enat 1
        using just[OF Suc(3) False Suc(4)] by blast
      then have nsinf:num (start G e) ≠ ∞
        by (metis Suc(2) enat.simps(3) enat-1 plus-enat-simps(2))
      then have ns:enat n = num (start G e)
        using e-assms(3) Suc(2) by force
      have dist (start G e ) ≠ − ∞
        by (rule Suc(1) [OF ns start-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)]
    start-value-check[OF s-in-G] start-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 ShortestPathsNeg
```

**imports**
*ShortestPath*

**begin**

# 2 Shortest Path (general edge costs)

**locale** *shortest-paths-locale-step1* =
**fixes** $G$ :: $('a, 'b)$ *pre-graph* (**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*: *pseudo-digraph G*
**assumes** *s-assms*:
$s \in verts\ G$
$dist\ s \neq \infty$
*parent-edge s = None*
$num\ s = 0$
**assumes** *parent-num-assms*:
$\bigwedge v.\ [\![v \in verts\ G;\ v \neq s;\ dist\ v \neq \infty]\!] \Longrightarrow$
$(\exists\, e \in edges\ G.\ parent\text{-}edge\ v = Some\ e\ \wedge$
$target\ G\ e = v \wedge dist\ (start\ G\ e) \neq \infty\ \wedge$
$num\ v = num\ (start\ G\ e) + 1)$
**assumes** *noPedge*: $\bigwedge e.\ e{\in}edges\ G \Longrightarrow$
$dist\ (start\ G\ e) \neq \infty \Longrightarrow dist\ (target\ G\ e) \neq \infty$

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

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

**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*:
$\bigwedge e.\ e \in edges\ G \Longrightarrow dist\ (start\ G\ e) = -\ \infty \Longrightarrow \forall\ r.\ dist\ (target\ G\ e) \neq ereal$
$r$

**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* (*start 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*:
$v = s \vee dist\ v = \infty \vee v \notin verts\ G \Longrightarrow pwalk\ v = []$
$v \neq s \Longrightarrow dist\ v \neq \infty \Longrightarrow v \in verts\ G \Longrightarrow pwalk\ v = pwalk\ (start\ G\ (the\ (parent\text{-}edge\ v)))\ @\ [the\ (parent\text{-}edge\ v)]$
**by** *auto*

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

**locale** *shortest-paths-locale-step3* =
*shortest-paths-locale-step2* +
**fixes** $C :: ('a \times ('b\ ewalk))\ set$
**assumes** *C-se*:
$C \subseteq \{(u, p).\ dist\ u \neq \infty \wedge ewalk\ u\ p\ u \wedge ewalk\text{-}cost\ c\ p < 0\}$
**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*:
$\bigwedge e.\ e \in edges\ G \Longrightarrow dist\ (start\ G\ e) = -\infty \Longrightarrow \forall\ r.\ dist\ (target\ G\ e) \neq ereal\ r$

**lemma** (**in** *wellformed-graph*) *edge-is-vwalk*:
**fixes** $e :: 'b$
**fixes** $u\ v :: 'a$
**assumes** $e \in edges\ G$
**assumes** $edge\text{-}to\text{-}ends\ G\ e = (u, v)$
**shows** $vwalk\ [u, v]\ G$
**proof** (*intro vwalkI*)
**show** $set\ [u,v] \subseteq verts\ G$
**using** *assms start-in-verts target-in-verts*
**unfolding** *edge-to-ends-def* **by** *auto*
**next**
**have** $(u, v) \in edges\text{-}ends\ G$
**unfolding** *edges-ends-def*
**using** *assms* **by** *force*

**moreover**
  **have** *vwalk-edges* $[u,\ v] = [(u,\ v)]$
    **using** *vwalk-edges.simps*(*2*)
    *vwalk-edges.simps*(*3*) **by** *simp*
**ultimately**
  **show** *set* (*vwalk-edges* $[u,\ v]$) $\subseteq$ *edges-ends G*
    **by** *simp*
**qed** *simp*

**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*) *vwalk-s*:
**fixes** $v :: 'a$
**assumes** $v = s$
**shows** $\exists$ *p*. *vwalk p G* $\wedge$ *hd p* $= s \wedge$ *last p* $= v$
 **by** (*metis assms s-assms*(*1*) *path-self hd.simps last-ConsL pathE*)

**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** $\exists$ *p e*. *vwalk p G* $\wedge$ *hd p* $= s \wedge$ *last p* $=$ *start G e* $\wedge$
        $e \in$ *edges G* $\wedge$ *target G e* $= v \wedge$ *dist* (*start G e*) $\neq \infty \wedge$
        *parent-edge* $v =$ *Some e* $\wedge$ *num* $v =$ *num* (*start G e*) $+ 1$
**using** *assms*
**proof**(*induct num* $v - 1$ *arbitrary* : *v*)
**case** *0*
  **obtain** *e* **where** *ee*:
    $e \in$ *edges G target G e* $= v$ *dist* (*start G e*) $\neq \infty$
    *parent-edge* $v =$ *Some e num* $v =$ *num* (*start G e*) $+ 1$
    **using** *parent-num-assms*[*OF 0*(*2*−*4*)] **by** *fast*
  **have** *start G e* $= s$
    **using** *num-s-is-min*[*OF start-in-verts* [*OF ee*(*1*)] - *ee*(*3*)]
    *ee*(*5*) *0*(*1*) **by** *auto*
  **thus** *?case* **using** *ee vwalk-s* **by** *blast*
**next**
**case** (*Suc n'*)
  **obtain** *e* **where** *ee*:
    $e \in$ *edges G target G e* $= v$ *dist* (*start G e*) $\neq \infty$
    *parent-edge* $v =$ *Some e num* $v =$ *num* (*start G e*) $+ 1$
    **using** *parent-num-assms*[*OF Suc*(*3*−*5*)] **by** *fast*
  **then have** *ss*: *start G e* $\neq s$
    **using** *num-s-is-min start-in-verts*

*Suc*(*2*) *s-assms*(*4*) **by** *force*
**have** *nst*: $n' = num\ (start\ G\ e) - 1$
**using** *ee*(*5*) *Suc*(*2*) **by** *presburger*
**obtain** $p'$ $e'$ **where** *sa*:
*vwalk* $p'$ *G* *hd* $p' = s$ *last* $p' = start\ G\ e'$
$e' \in edges\ G$ *target* $G\ e' = start\ G\ e$ *dist* $(start\ G\ e') \neq \infty$
**using** *Suc*(*1*)[*OF nst start-in-verts*[*OF ee*(*1*)] *ss ee*(*3*)] **by** *blast*
**then have** *vwalk* [*start G e*′, *start G e*] *G*
**using** *edge-is-vwalk* **unfolding** *edge-to-ends-def* **by** *fast*
**then have** $\exists p.\ vwalk\ p\ G \wedge hd\ p = s \wedge last\ p = start\ G\ e$
**by** (*metis hd.simps last-ConsL sa*(*1*−*3*)
*last-ConsR vwalkE list.simps*(*3*) *joinableI*
*vwalk-join-hd vwalk-join-vwalk vwalk-join-last*)
**thus** *?case* **using** *ee* **by** *simp*
**qed**

**lemma** (**in** *shortest-paths-locale-step1*) *path-from-root-Vr*:
**fixes** *v* :: ′*a*
**assumes** $v \in verts\ G$
**assumes** *dist* $v \neq \infty$
**shows** $s \rightarrow* G\ v$
**proof**(*cases* $v = s$)
**case** *True* **thus** *?thesis*
**unfolding** *reachable-def* **using** *vwalk-s* **by** *simp*
**next**
**case** *False*
**obtain** *p e* **where**
*pe*: $vwalk\ p\ G \wedge hd\ p = s \wedge last\ p = start\ G\ e \wedge$
$e \in edges\ G \wedge target\ G\ e = v \wedge dist\ (start\ G\ e) \neq \infty \wedge$
$parent\text{-}edge\ v = Some\ e \wedge num\ v = num\ (start\ G\ e) + 1$
**using** *path-from-root-Vr-ex*[*OF assms*(*1*) *False assms*(*2*)] **by** *blast*
**then have** *vwalk* [*start G e*, *v*] *G*
**using** *edge-is-vwalk* **unfolding** *edge-to-ends-def* **by** *fast*
**thus** *?thesis* **unfolding** *reachable-def*
**by** (*metis hd.simps last-ConsL pe*
*last-ConsR vwalkE list.simps*(*3*) *joinableI*
*vwalk-join-hd vwalk-join-vwalk vwalk-join-last*)
**qed**

**lemma** (**in** *shortest-paths-locale-step1*) $\mu$*-V-less-inf*:
**fixes** *v* :: ′*a*
**assumes** $v \in verts\ G$
**assumes** *dist* $v \neq \infty$
**shows** $\mu\ c\ s\ v \neq \infty$
**using** *assms path-from-root-Vr* $\mu$*-reach-conv* **by** *force*

**lemma** (**in** *shortest-paths-locale-step2*) *enum-not0*:
**assumes** $v \in verts\ G$
**assumes** $v \neq s$

```
  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:ewalk s [] s
    using s-assms(1) unfolding ewalk-def by simp
  have mu: μ c s s = ereal 0
    using min-cost-le-walk-cost[OF ews, where c=c]
    ewalk-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-ewalk:
  fixes v :: 'a
  assumes v ∈ verts G
  assumes dist v ≠ ∞
  shows ewalk s (pwalk v) v
proof (cases v=s)
case True
  thus ?thesis
    using assms pwalk.simps[where v=v]
    ewalk-empty-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 = start G ?e
    assume ewu: ¬ (v = s ∨ dist v = ∞ ∨ v ∉ verts G) ⟹
              ?u ∈ verts G ⟹ dist ?u ≠ ∞ ⟹
              ewalk s (pwalk ?u) ?u
    assume vG: v ∈ verts G
    assume dv: dist v ≠ ∞
    thus ewalk s (pwalk v) v
    proof (cases v = s ∨ dist v = ∞ ∨ v ∉ verts G)
    case True
```

**thus** *?thesis*
**using** *pwalk.simps vG dv*
*ewalk-empty-iff* **by** *fastforce*
**next**
**case** *False*
**obtain** *e* **where** *ee*:
$e \in edges\ G$
$parent\text{-}edge\ v = Some\ e$
$target\ G\ e = v$
$dist\ (start\ G\ e) \neq \infty$
**using** *parent-num-assms False* **by** *blast*
**hence** $ewalk\ s\ (pwalk\ ?u)\ ?u$
**using** *ewu*[*OF False*] *start-in-verts* **by** *simp*
**hence** $ewalk\ s\ (pwalk\ (start\ G\ e)\ @\ [e])\ v$
**by** (*metis ee*(*1*−*3*) *ewalk-Cons-iff ewalk-empty-iff ewalk-ewalk-joinI the.simps vG*)
**thus** *?thesis* **using** *False ee*(*2*) *pwalk.simps* **by** *auto*
**qed**
**qed**
**qed**

**lemma** (**in** *shortest-paths-locale-step3*) *μ-ninf*:
**fixes** $v :: 'a$
**assumes** $v \in verts\ G$
**assumes** $dist\ v = -\ \infty$
**shows** $\mu\ c\ s\ v = -\ \infty$
**proof** −
**have** $ewalk\ s\ (pwalk\ v)\ v$
**using** *pwalk-ewalk assms* **by** *force*
**moreover**
**obtain** *w* **where** *ww*: $w \in fst\ `\ C \cap pwalk\text{-}verts\ v$
**using** *int-neg-cyc*[*OF assms*] **by** *blast*
**then obtain** *q* **where**
$ewalk\ w\ q\ w$
$ewalk\text{-}cost\ c\ q < 0$
**using** *C-se* **by** *auto*
**moreover**
**have** $w \in set\ (ewalk\text{-}verts\ s\ (pwalk\ v))$
**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-*$\mu$[*OF assms*] **by** *simp*
**next**
**show** *dist* $v = \infty \Longrightarrow$ *dist* $v = \mu\ c\ s\ v$
**using** $\mu$*-V-less-inf* [*OF assms*]
*dist-le-*$\mu$[*OF assms*] **by** *simp*
**next**
**show** *dist* $v = -\infty \Longrightarrow$ *dist* $v = \mu\ c\ s\ v$
**using** $\mu$*-ninf* [*OF assms*] **by** *simp*
**qed**

**end**