The Königsberg Bridge Problem and the Friendship
Theorem

Wenda Li
May 27, 2015

Abstract

This development provides a formalization of undirected graphs
and simple graphs, which are based on Benedikt Nordhoff and Peter
Lammich’s simple formalization of labelled directed graphs [4] in the
archive. Then, with our formalization of graphs, we have shown both
necessary and sufficient conditions for Eulerian trails and circuits [2]
as well as the fact that the Königsberg Bridge problem does not have
a solution. In addition, we have also shown the Friendship Theorem
in simple graphs[1, 3].

Contents

1 Undirected Multigraph and undirected trails 2
2 Degrees and related properties 2
3 Connectivity 30
4 Adjacent nodes 45
5 Undirected simple graph 46
6 Definition of Eulerian trails and circuits 49
7 Necessary conditions for Eulerian trails and circuits 49
8 Specific case of the Königsberg Bridge Problem 51
9 Sufficient conditions for Eulerian trails and circuits 53
10 Common steps 72
theory MoreGraph imports Complex-Main ../Dijkstra-Shortest-Path/Graph
begin

1 Undirected Multigraph and undirected trails

locale valid-unMultigraph = valid-graph G for G::(′v,′w) graph+
  assumes corres[simp]: (v,w,u') ∈ edges G ←→ (v',w,v) ∈ edges G
  and  no-id[simp]: (v,w,v) /∈ edges G

fun (in valid-unMultigraph) is-trail :: ′v ⇒ (′v,′w) path ⇒ bool where
  is-trail v [] v' ←→ v=v' ∧ v' ∈ V |
  is-trail v ((v1,w,v2)#ps) v' ←→ v=v1 ∧ (v1,w,v2)∈ E ∧
  (v1,w,v2) /∈ set ps ∧ (v2,w,v1) /∈ set ps ∧ is-trail v2 ps v'    

2 Degrees and related properties

definition degree :: ′v ⇒ (′v,′w) graph ⇒ nat where
  degree v g≡ card({ e. e ∈ edges g ∧ fst e=v})

definition odd-nodes-set :: (′v,′w) graph ⇒ ′v set where
  odd-nodes-set g≡ {v. v ∈ nodes g ∧ odd(degree v g)}

definition num-of-odd-nodes :: (′v,′w) graph ⇒ nat where
  num-of-odd-nodes g≡ card( odd-nodes-set g)

definition num-of-even-nodes :: (′v,′w) graph ⇒ nat where
  num-of-even-nodes g≡ card( {v. v ∈ nodes g ∧ even(degree v g)})

definition del-unEdge where del-unEdge v e v' g≡ []
  nodes = nodes g, edges = edges g - {(v,e,v'),(v',e,v)} []

definition rev-path :: (′v,′w) path ⇒ (′v,′w) path where
  rev-path ps ≡ map (λ(a,b,c).(c,b,a)) (rev ps)

definition rem-unPath :: (′v,′w) path ⇒ (′v,′w) graph ⇒ (′v,′w) graph where
  rem-unPath [] g= g|
  rem-unPath ((v,w,v')#ps) g= rem-unPath ps (del-unEdge v w v' g)

lemma del-undirected: del-unEdge v e v' g = delete-edge v e v' (delete-edge v e v'
  g)
  unfolding del-unEdge-def delete-edge-def by auto

lemma delete-edge-sym: del-unEdge v e v' g = del-unEdge v' e v g

2
unfolding del-unEdge-def by auto

lemma del-unEdge-valid(simp); assumes valid-unMultigraph g shows valid-unMultigraph (del-unEdge v e v' g)
proof –
interpret valid-unMultigraph g by fact
show ?thesis
unfolding del-unEdge-def
by unfold-locales (auto dest: E-validD)
qed

lemma set-compre-diff:{x ∈ A − B. P x}={x ∈ A. P x} − {x ∈ B . P x} by blast
lemma set-compre-subset: B ⊆ A → {x ∈ B. P x} ⊆ {x ∈ A. P x} by blast

lemma del-edge-undirected-degree-plus: finite (edges g) → (v,v',v) ∈ edges g → (v',e,v) ∈ edges g → degree v (del-unEdge v e v' g) + 1 = degree v g
proof –
assume assms: finite (edges g) (v,v',v) ∈ edges g (v',e,v) ∈ edges g
have degree v (del-unEdge v e v' g) + 1 = card ({ea ∈ edges g − {(v, e, v'), (v', e, v)}, fst ea = v}) + 1
unfolding del-unEdge-def degree-def by simp
also have ...=card ({ea ∈ edges g. fst ea = v} − {ea ∈ {(v, e, v'), (v', e, v)}}.
  fst ea = v}) + 1
by (metis set-compre-diff)
also have ...=card ({ea ∈ edges g. fst ea = v}) − card({ea ∈ {(v, e, v'), (v', e, v)}.
  fst ea = v}) + 1
proof –
have {(v, e, v'), (v', e, v)} ⊆ edges g using ⟨(v,e,v') ∈ edges g ⟩ ⟨(v',e,v) ∈ edges g ⟩
  by auto
hence {ea ∈ {(v, e, v'), (v', e, v)}. fst ea = v} ⊆ {ea ∈ edges g. fst ea = v} by auto
moreover have finite {ea ∈ {(v, e, v'), (v', e, v)}}. fst ea = v by auto
ultimately have card ({ea ∈ edges g. fst ea = v} − {ea ∈ {(v, e, v'), (v', e, v)}.
  fst ea = v})=card {ea ∈ edges g. fst ea = v} − card {ea ∈ {(v, e, v'),
  (v', e, v)}}.
  fst ea = v}
using card-Diff-subset by blast
thus ?thesis by auto
qed
also have ...=card ({ea ∈ edges g. fst ea = v})
proof –
have {ea ∈ {(v, e, v'), (v', e, v)}. fst ea = v}={v,e,v'} by auto
hence card {ea ∈ {(v, e, v'), (v', e, v)}. fst ea = v} = 1 by auto
moreover have card {ea ∈ edges g. fst ea = v}≠0
ultimately show \(?thesis\) by arith

qed

finally have degree \(v\) \((\text{del-unEdge } v e v' g) + 1 = \text{card}\{ea \in \text{edges } g. \text{fst } ea = v\}\).

thus \(?thesis\) unfolding degree-def.

qed

lemma \text{del-edge-undirected-degree-plus}: finite \((\text{edges } g) \implies (v,e,v') \in \text{edges } g \implies (v',e,v) \in \text{edges } g \implies \text{degree } v' (\text{del-unEdge } v e v' g) + 1 = \text{degree } v' g\)

by (metis \text{del-edge-undirected-degree-plus delete-edge-sym})

lemma \text{del-edge-undirected-degree-minus} [simp]: finite \((\text{edges } g) \implies (v,e,v') \in \text{edges } g \implies (v',e,v) \in \text{edges } g \implies \text{degree } v' (\text{del-unEdge } v e v' g) = \text{degree } v' g - (1::nat)

using \text{del-edge-undirected-degree-plus} by (metis add-diff-cancel-left add.commute)

lemma \text{del-edge-undirected-degree-minus} [simp]: finite \((\text{edges } g) \implies (v,e,v') \in \text{edges } g \implies (v',e,v) \in \text{edges } g \implies \text{degree } v' (\text{del-unEdge } v e v' g) = \text{degree } v' g - (1::nat)

by (metis \text{del-edge-undirected-degree-minus delete-edge-sym})

lemma \text{del-unEdge-com}: \text{del-unEdge } v w v' (\text{del-unEdge } n e n' g) = \text{del-unEdge } n e n' (\text{del-unEdge } v w v' g)

unfolding \text{del-unEdge-def} by auto

lemma \text{rem-unPath-com}: \text{rem-unPath } ps (\text{del-unEdge } v w v' g) = \text{del-unEdge } v w v' (\text{rem-unPath } ps g)

proof (induct ps arbitrary: g)

case Nil

thus \(?case\) by (metis \text{rem-unPath.simps}(1))

next

case (Cons a ps')

thus \(?case\) using \text{del-unEdge-com}

by (metis prod-cases3 \text{rem-unPath.simps}(1) \text{rem-unPath.simps}(2))

qed

lemma \text{rem-unPath-valid} [intro]:
valid-unMultigraph \(g\) \implies valid-unMultigraph \((\text{rem-unPath } ps g)\)

proof (induct ps)

case Nil

thus \(?case\) by simp

next

case (Cons x xs)

thus \(?case\)

proof –
lemma (in valid-unMultigraph) degree-frame:
  assumes finite (edges G) x \notin \{v, v'\}
  shows degree x (del-unEdge v w v' G) = degree x G (is ?L=?R)
proof (cases (v,v',v') \in edges G)
  case True
  have ?L=card({(e, e \in edges G - \{(v,w,v'),(v',w,v)\} \land fst e=x})
    by (simp add:del-unEdge-def degree-def)
  also have ...=card({(e, e \in edges G \land fst e=x}-\{e, e \in\{(v,w,v'),(v',w,v)\} \land fst e=x})
  qed
  qed

lemma (in valid-unMultigraph) rev-path-def unfolding rev-path-double by simp
lemma rev-path-append[simp]: rev-path (xs@ys) = (rev-path ys) @ (rev-path xs)
unfolding rev-path-def rev-append by auto
lemma rev-path-double[simp]: rev-path(rev-path xs)=xs
unfolding rev-path-def by (induct xs,auto)

lemma del-UnEdge-node[simp]: v\in nodes (del-unEdge u e v' G) \leftrightarrow v\in nodes G
  by (metis del-unEdge-def select-convs)
lemma [intro!]: finite (edges G) \implies finite (edges (del-unEdge u e u' G))
  by (metis del-unEdge-def finite-Diff select-convs(2))

lemma [intro!]: finite (nodes G) \implies finite (nodes (del-unEdge u e u' G))
  by (metis del-unEdge-def select-convs(1))

lemma [intro!]: finite (edges G) \implies finite (edges (rem-unPath ps G))
proof (induct ps arbitrary:G)
  case Nil
  thus ?case by simp
next
  case (Cons x xs)
  hence finite (edges (rem-unPath (x # xs) G)) = finite (edges (del-unEdge (fst x) (fst (snd x)) (snd (snd x)) (rem-unPath xs G)))
    by (metis rem-unPath.simps(2) rem-unPath-com surjective-pairing)
  also have ...=finite (edges (rem-unPath xs G))
    using del-unEdge-def
    by (metis finite.emptyI finite-Diff2 finite-Diff-insert select-convs(2))
  also have ...=True using Cons by auto
  finally have ?case = True.
  thus ?case by simp
qed

lemma del-UnEdge-frame[intro]:
  x \in edges g \implies x \neq (v, e, v') \implies x \notin edges (del-unEdge v e v' g)
unfolding del-unEdge-def by auto

lemma [intro!]: finite (nodes G) \implies finite (odd-nodes-set G)
  by (metis (lifting) mem-Collect-eq odd-nodes-set-def rev-finite-subset subsetI)

lemma [simp]: nodes (del-unEdge u e u' G)=nodes G
  by (metis del-unEdge-def select-convs(1))

lemma [simp]: nodes (rem-unPath ps G) = nodes G
proof (induct ps)
  case Nil
  show ?case by simp
next
  case (Cons x xs)
  have nodes (rem-unPath (x # xs) G)=nodes (del-unEdge (fst x) (fst (snd x)) (snd (snd x)) (rem-unPath xs G))
    by (metis rem-unPath.simps(2) rem-unPath-com surjective-pairing)
  also have ...=nodes (rem-unPath xs G) by auto
  also have ...=nodes G using Cons by auto
  finally show ?case.
qed

lemma [intro!]: finite (nodes G) \implies finite (nodes (rem-unPath ps G)) by auto
lemma in-set-rev-path[simp]: \( (v', w, v) \in \text{set (rev-path } ps) \iff (v, w, v') \in \text{set } ps \)

proof (induct ps)
  case Nil
  thus ?case unfolding rev-path-def by auto

next
  case (Cons x xs)
  obtain x1 x2 x3 where x:=x1,x2,x3 by (metis prod-cases3)
  have set (rev-path (x # xs)) = set ((rev-path xs)@[x3,x2,x1])
    unfolding rev-path-def
    using x by auto
  also have ... = set (rev-path xs) ∪ \{x3,x2,x1\} by auto
  finally have set (rev-path (x # xs)) = set (rev-path xs) ∪ \{x3,x2,x1\}.
  moreover have set (x # xs) = set xs ∪ \{x1,x2,x3\}
  by (metis List.set-simps(2) insert-is-Un sup-commute x)
  ultimately show ?case using Cons by auto

qed

lemma rem-unPath-edges:
  edges(rem-unPath ps G) = edges G - (set ps ∪ set (rev-path ps))
proof (induct ps)
  case Nil
  show ?case unfolding rev-path-def by auto

next
  case (Cons x xs)
  obtain x1 x2 x3 where x:=x1,x2,x3 by (metis prod-cases3)
  hence edges(rem-unPath (x # xs) G) = edges(del-unEdge x1 x2 x3 (rem-unPath xs G))
    by (metis rem-unPath.simps(2) rem-unPath-com)
  also have ... = edges(rem-unPath xs G) - \{x1,x2,x3\} ∪ \{x3,x2,x1\}
    by (metis del-unEdge-def select-consvs(2))
  also have ... = edges G - (set xs ∪ set (rev-path xs)) - \{x1,x2,x3\} ∪ \{x3,x2,x1\}
    by (metis Cons.hyps)
  also have ... = edges G - (set (x # xs) ∪ set (rev-path (x # xs)))
  proof
    have set (rev-path xs) ∪ \{x3,x2,x1\} = set (rev-path xs)@[x3,x2,x1]
      by (metis List.set-simps(2) empty-set set-append)
    also have ... = set (rev-path (x # xs)) unfolding rev-path-def using x by auto
    finally have set (rev-path xs) ∪ \{x3,x2,x1\} = set (rev-path (x # xs)) .
    moreover have set xs ∪ \{x1,x2,x3\} = set (x # xs)
      by (metis List.set-simps(2) insert-is-Un sup-commute x)
    moreover have edges G - (set xs ∪ set (rev-path xs)) - \{x1,x2,x3\} ∪ \{x3,x2,x1\}
      = edges G - ((set xs ∪ \{x1,x2,x3\}) ∪ (set (rev-path xs) ∪ \{x3,x2,x1\})))
      by auto
    ultimately show ?thesis by auto
  qed
finally show ?case .

7
lemma rem-unPath-graph [simp]:
rem-unPath (rev-path ps) G = rem-unPath ps G
proof -
  have nodes (rem-unPath (rev-path ps) G) = nodes (rem-unPath ps G)
    by auto
  moreover have edges (rem-unPath (rev-path ps) G) = edges (rem-unPath ps G)
    proof -
      have set (rev-path ps) ∪ set (rev-path (rev-path ps)) = set ps ∪ set (rev-path ps)
        by auto
      thus ?thesis by (metis rem-unPath-edges)
    qed
  ultimately show ?thesis by auto
qed

lemma distinct-rev-path [simp]: distinct (rev-path ps) ←→ distinct ps
proof (induct ps)
  case Nil
  show ?case by auto
next
  case (Cons x xs)
  obtain x1 x2 x3 where x = (x1, x2, x3) by (metis prod-cases3)
  hence distinct (rev-path (x # xs)) = distinct ((rev-path xs) @ [(x3, x2, x1)])
    unfolding rev-path-def by auto
  also have ... = (distinct (rev-path xs) ∧ (x3, x2, x1) ∉ set (rev-path xs))
    by (metis distinct.simps(2) distinct1_rotate rotate1.simps(2))
  also have ... = distinct (x#xs)
    by (metis Cons.hyps distinct.simps(2) in-set-rev-path)
  finally have distinct (rev-path (x # xs)) = distinct (x#xs).
  thus ?case.
qed

lemma (in valid-unMultigraph) is-path-rev: is-path v' (rev-path ps) \ v ←→ is-path v ps v'
proof (induct ps arbitrary: v)
  case Nil
  show ?case by auto
next
  case (Cons x xs)
  obtain x1 x2 x3 where x = (x1, x2, x3) by (metis prod-cases3)
  hence is-path v' (rev-path (x # xs)) = is-path v' ((rev-path xs) @ [(x3, x2, x1)])
    unfolding rev-path-def by auto
  also have ... = (is-path v' (rev-path xs) x3 ∧ (x3, x2, x1) ∈ E ∧ is-path x1 \ v)
    by auto
  also have ... = (is-path x3 xs v' ∧ (x3, x2, x1) ∈ E ∧ is-path x1 \ v) using Cons.hyps

by auto
also have \ldots = is-path v (x#xs) v'
  by (metis corres is-path.simps(1) is-path.simps(2) is-path-memb x)
finally have is-path v' (rev-path (x # xs)) v = is-path v (x#xs) v'.
thus \?case .
qed

lemma (in valid-unMultigraph) singleton-distinct-path [intro]:
  (v,w,v')\in E \implies is-trail v [(v,w,v')] v'
by (metis E-validD(2) all-not-in-conv is-trail.simps set-empty)

lemma (in valid-unMultigraph) is-trail-path:
  is-trail v ps v' \iff is-path v ps v' \land distinct ps \land (set ps \cap set (rev-path ps) = \{\})
proof (induct ps arbitrary:v)
case Nil
  show \?case by auto
next
case (Cons x xs)
  obtain x1 x2 x3 where x: x=(x1,x2,x3) by (metis prod-cases3)
hence is-trail v (x#xs) v' = (v=x1 \land (x1,x2,x3)\in E \land
  (x1,x2,x3)\notin set xs \land (x3,x2,x1)\notin set xs \land is-trail x3 xs v')
  by (metis is-trail.simps(2))
also have \ldots = (v=x1 \land (x1,x2,x3)\in E \land (x1,x2,x3)\notin set xs \land (x3,x2,x1)\notin set xs
  \land is-path x3 xs v' \land distinct xs \land (set xs \cap set (rev-path xs) = \{\}))
  using Cons.hyps by auto
also have \ldots = (is-path v (x#xs) v' \land (x1,x2,x3) \neq (x3,x2,x1) \land (x1,x2,x3)\notin set xs
  \land (x3,x2,x1)\notin set xs \land distinct xs \land (set xs \cap set (rev-path xs) = \{\}))
  by (metis append-Nil is-path.simps(1) is-path.simps(2) is-path-split' no-id x)
also have \ldots = (is-path v (x#xs) v' \land (x1,x2,x3) \neq (x3,x2,x1) \land (x1,x2,x3)\notin set xs
  \land (x3,x2,x1)\notin set xs \land (set xs \cap set (rev-path xs) = \{\}))
  by (metis (full-types) distinct.simps(2) x)
also have \ldots = (is-path v (x#xs) v' \land (x1,x2,x3) \neq (x3,x2,x1) \land distinct (x#xs)
  \land (x3,x2,x1)\notin set xs \land (set xs \cap set (rev-path (x#xs) = \{\}))
proof -
  have set (rev-path (x#xs)) = set ((rev-path xs)@[((x3,x2,x1)]) using x by auto
  also have \ldots = set (rev-path xs) \cup \{(x3,x2,x1)\} by auto
  finally have set (rev-path (x#xs)) = set (rev-path xs) \cup \{(x3,x2,x1)\}.
  thus \?thesis by blast
qed
also have \ldots = (is-path v (x#xs) v' \land distinct (x#xs) \land (set (x#xs) \cap set (rev-path
  (x#xs)) = \{\}))
proof -
have \((x3, x2, x1) \notin \text{set } xs \leftrightarrow (x1, x2, x3) \notin \text{set } (\text{rev-path } xs)\) using in-set-rev-path by auto

moreover have set (\text{rev-path } (x\#xs)) = set (\text{rev-path } xs) \cup \{(x3, x2, x1)\}

unfolding \text{rev-path-def} using x by auto

ultimately have (x1, x2, x3) \neq (x3, x2, x1) \land (x3, x2, x1) \notin \text{set } xs

\leftrightarrow (x1, x2, x3) \notin \text{set } (\text{rev-path } (x\#xs))\) by blast

thus \(?thesis\)

by (metis (mono-tags) Int-iff Int-insert-left-if0 List.set-simps(2) empty-iff insertII x)

qed

finally have is-trail v (x\#xs) \iff (is-path v (x\#xs) v' \land \text{distinct } (x\#xs)

\land (\text{set } (x\#xs) \cap \text{set } (\text{rev-path } (x\#xs)) )=\{\})\).

thus \(?case\).

qed

lemma (in valid-unMultigraph) is-trail-rev:

is-trail v' (\text{rev-path } ps) v \leftrightarrow is-trail v ps v'

using \text{rev-path-append} is-trail-path is-path-rev distinct-rev-path

by (metis \text{In-commute} distinct-append)

lemma (in valid-unMultigraph) is-trail-intro[intro]:

is-trail v' ps v \implies is-path v' ps v by (induct ps arbitrary; v',auto)

lemma (in valid-unMultigraph) is-trail-split:

is-trail v (p1@p2) v' \implies (\exists u. is-trail v p1 u \land is-trail u p2 v')

apply (induct p1 arbitrary; v',auto)

apply (metis is-trail-intro is-path-memb)

done

lemma (in valid-unMultigraph) is-trail-split':is-trail v (p1@\langle u,w,u'\rangle#p2) v' 

\implies is-trail v p1 u \land (u,w,u')\in E \land is-trail u' p2 v'

by (metis is-trail.split simp)

lemma (in valid-unMultigraph) distinct-elim[simp]:

assumes is-trail v ((v1, w, v2)#ps) v'

shows (v1, w, v2)\in edges (\text{rem-unPath } ps G) \leftrightarrow (v1, w, v2)\in E

proof

assume (v1, w, v2) \in edges (\text{rem-unPath } ps G)

thus (v1, w, v2) \in E by (metis assms is-trail.split simp)

next

assume (v1, w, v2) \in E

have (v1, w, v2)\notin \text{set } ps \land (v2, w, v1)\notin \text{set } ps by (metis assms is-trail.split simp)

hence (v1, w, v2)\notin \text{set } ps \land (v1, w, v2)\notin \text{set } (\text{rev-path } ps) by simp

hence (v1, w, v2)\notin \text{set } ps \cup \text{set } (\text{rev-path } ps) by simp

hence (v1, w, v2)\in edges G \land (\text{set } ps \cup \text{set } (\text{rev-path } ps))

using (v1, w, v2) \in E by auto

thus (v1, w, v2) \in edges (\text{rem-unPath } ps G)

by (metis \text{rem-unPath-edges})

qed
lemma distinct-path-subset:
assumes valid-unMultigraph G1 valid-unMultigraph G2 edges G1 ⊆ edges G2
nodes G1 ⊆ nodes G2
assumes distinct-G1:valid-unMultigraph.is-trail G1 v ps v'
sows valid-unMultigraph.is-trail G2 v ps v' using distinct-G1
proof (induct ps arbitrary:v)
case Nil
hence v=v'\land v'\in nodes G1
by (metis (full-types) assms(1) valid-unMultigraph.is-trail.simps(1))
hence v=v'\land v'\in nodes G2 using (nodes G1 ⊆ nodes G2) by auto
thus ?case by (metis assms(2) valid-unMultigraph.is-trail.simps(1))
next
case (Cons x xs)
obtain x1 x2 x3 where x=x1 x2 x3 by (metis prod-cases3)
hence valid-unMultigraph.is-trail G1 x3 xs v'
by (metis Cons.prems assms(1) valid-unMultigraph.is-trail.simps(2))
hence valid-unMultigraph.is-trail G2 x3 xs v' using Cons by auto
moreover have x\in edges G1
by (metis Cons.prems assms(1) valid-unMultigraph.is-trail.simps(2))
hence x\in edges G2 using (edges G1 ⊆ edges G2) by auto
moreover have v=x1\land (x1 x2 x3)\notin set xs \land (x3 x2 x1)\notin set xs
by (metis Cons.prems assms(1) valid-unMultigraph.is-trail.simps(2))
hence v=x1 (x1 x2 x3)\notin set xs (x3 x2 x1)\notin set xs by auto
ultimately show ?case by (metis assms(2) valid-unMultigraph.is-trail.simps(2))
qed

lemma (in valid-unMultigraph) distinct-path-intro':
assumes valid-unMultigraph.is-trail (rem-unPath p G) v ps v'
sows is-trail v ps v'
proof –
have valid:valid-unMultigraph (rem-unPath p G)
using rem-unPath-valid[OF valid-unMultigraph.axioms,of p] by auto
moreover have nodes (rem-unPath p G) ⊆ V by auto
moreover have edges (rem-unPath p G) ⊆ E
using rem-unPath-edges by auto
ultimately show ?thesis
using distinct-path-subset[of rem-unPath p G G] valid-unMultigraph.axioms assms
by auto
qed

lemma (in valid-unMultigraph) distinct-path-intro:
assumes valid-unMultigraph.is-trail (del-unEdge x1 x2 x3 G) v ps v'
sows is-trail v ps v'
by (metis (full-types) assms distinct-path-intro' rem-unPath.simps(1)
rem-unPath.simps(2))
lemma (in valid-unMultigraph) distinct-elim-rev[simp]:
assumes is-trail v ((v1,w,v2)#ps) v' 
shows (v2,w,v1) ∈ edges (rem-unPath ps G) ⟷ (v2,w,v1) ∈ E
proof –
  have valid-unMultigraph (rem-unPath ps G) using valid-unMultigraph-axioms 
    by auto
  hence (v2,w,v1) ∈ edges (rem-unPath ps G) ⟷ (v1,w,v2) ∈ edges (rem-unPath ps G)
    by (metis valid-unMultigraph.corres)
moreover have (v2,w,v1) ∈ E ⟷ (v1,w,v2) ∈ E using corres by simp
ultimately show ?thesis using distinct-elim by (metis assms)
qed

lemma (in valid-unMultigraph) del-UnEdge-even: 
assumes (v,w,v') ∈ E finite E
shows v ∈ odd-nodes-set (del-unEdge v w v' G) ⟷ even (degree v G)
proof –
  show ?thesis unfolding odd-nodes-set-def by auto
qed

lemma (in valid-unMultigraph) del-UnEdge-even':
assumes (v,w,v') ∈ E finite E
shows v' ∈ odd-nodes-set (del-unEdge v w v' G) ⟷ even (degree v' G)
proof –
  show ?thesis by (metis (full-types) assms corres del-UnEdge-even delete-edge-sym)
qed

lemma del-UnEdge-even-even:
assumes valid-unMultigraph G finite (edges G) finite (nodes G) (v, w, v') ∈ edges G
assumes parity-assms: even (degree v G) even (degree v' G)
shows num-of-odd-nodes (del-unEdge v w v' G) = num-of-odd-nodes G + 2
proof –
interpret G:valid-unMultigraph by fact
have v ∈ odd-nodes-set (del-unEdge v w v' G)
  by (metis G.del-UnEdge-even assms(1) parity-assms(1))
moreover have v' ∈ odd-nodes-set (del-unEdge v w v' G)
  by (metis G.del-UnEdge-even' assms(2) assms(4) parity-assms(2))
ultimately have extra-odd-nodes {v,v'} ⊆ odd-nodes-set (del-unEdge v w v' G)
  unfolding odd-nodes-set-def by auto
moreover have v ∉ odd-nodes-set G and v' ∉ odd-nodes-set G

using parity-assms unfolding odd-nodes-set-def by auto 

hence \( \{v, v'\} \cap \text{odd-nodes-set } G = \{\} \) by auto 

moreover have odd-nodes-set(del-unEdge v w v' G) \( \subseteq \text{odd-nodes-set } G \) 

proof 

fix \( x \) 

assume \( x\)-odd-set: \( x \in \text{odd-nodes-set (del-unEdge v w v' G)} \) 

hence degree \( x \) (del-unEdge v w v' G) = degree \( x \) G by (metis Diff-iff G.\text{degree-frame } \text{assms} (2)) 

hence odd\{(degree \( x \) G) using \( x\)-odd-set 

unfolding odd-nodes-set-def by auto 

moreover have \( x \) \( \in \text{nodes } G \) using \( x\)-odd-set unfolding odd-nodes-set-def 

by auto 

ultimately show \( x \) \( \in \text{odd-nodes-set } G \) unfolding odd-nodes-set-def by auto 

qed 

moreover have \( \text{odd-nodes-set } G \subseteq \text{odd-nodes-set (del-unEdge v w v' G)} \) 

proof 

fix \( x \) 

assume \( x\)-odd-set: \( x \in \text{odd-nodes-set G} \) 

hence \( x \notin \{v, v'\} \) \( \Rightarrow \) odd\{(degree \( x \) G) using \( x\)-odd-set 

unfolding odd-nodes-set-def by auto 

moreover have \( x \notin \{v, v'\} \) \( \Rightarrow x \in \text{odd-nodes-set (del-unEdge v w v' G)} \) 

using extra-odd-nodes by auto 

ultimately show \( x \) \( \in \text{odd-nodes-set (del-unEdge v w v' G)} \) by auto 

qed 

ultimately have \( \text{odd-nodes-set (del-unEdge v w v' G)} = \text{odd-nodes-set } G \cup \{v, v'\} \) 

by auto 

thus \( \text{num-of-odd-nodes (del-unEdge v w v' G)} = \text{num-of-odd-nodes } G + 2 \) 

proof – 

assume odd-nodes-set(del-unEdge v w v' G) = odd-nodes-set G \( \cup \{v, v'\} \) 

moreover have \( v \neq v' \) using G.\text{no-id } \{(v, w, v') \in \text{edges } G\} \text{ by auto} 

hence \( \text{card}\{v, v'\} = 2 \) by simp 

moreover have odd-nodes-set G \( \cap \{v, v'\} = \{\} \) 

using vv'-odd-disjoint by auto 

moreover have finite\{(odd-nodes-set G) 

by (metis (lifting) \text{assms} (3) \text{mem-Collect-eq odd-nodes-set-def rev-finite-subset subsetI}) 

moreover have finite \( \{v, v'\} \) by auto 

ultimately show \( ?\text{thesis}\) unfolding num-of-odd-nodes-def using card-Un-disjoint 

by metis 

qed 

qed 

lemma del-UnEdge-even-odd: 

assumes valid-unMultigraph G finite(edges G) finite(nodes G) \( (v, w, v') \in \text{edges G} \) 

assumes parity-assms: even \((\text{degree } v G) \) odd \((\text{degree } v' G) \) 

shows num-of-odd-nodes(del-unEdge v w v' G) = num-of-odd-nodes G
proof

interpret $G : \text{valid-unMultigraph}$ by fact

have odd-nodes-set-def $\mathbf{v} \in \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v')$ $G$
  by (metis $G$.del-UnEdge-even assms(2) assms(4) parity-assms(1))

have not-odd-$v'$: $v' \notin \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v')$ $G$
  by (metis $G$.del-UnEdge-even assms(2) assms(4) parity-assms(2))

have odd-nodes-set $(\text{del-unEdge} \ v \ w \ v') \cup \{v\} \subseteq \text{odd-nodes-set} \ G \cup \{v\}$

proof

fix $x$

assume x-prems: $x \in \text{odd-nodes-set} (\text{del-unEdge} \ v \ w \ v') \cup \{v\}$

have $x = v' \Rightarrow x \in \text{odd-nodes-set} \ G \cup \{v\}$
  using parity-assms
  by (metis (lifting) $G$.E-validD(2) Un-def assms(4) mem-Collect-eq odd-nodes-set-def)

moreover have $x = v \Rightarrow x \in \text{odd-nodes-set} \ G \cup \{v\}$
  by (metis insertI1 insert-is-Un sup-commute)

moreover have $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} (\text{del-unEdge} \ v \ w \ v')$ $G$
  using x-prems by auto

hence $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} \ G$ unfolding odd-nodes-set-def
  using $G$.degree-frame (finite (edges $G$)) by auto

hence $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} \ G \cup \{v\}$ by simp

ultimately show $x \in \text{odd-nodes-set} \ G \cup \{v\}$ by auto

qed

moreover have odd-nodes-set $G \cup \{v\} \subseteq \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v')$ $G$

proof

fix $x$

assume x-prems: $x \in \text{odd-nodes-set} \ G \cup \{v\}$

have $x = v' \Rightarrow x \in \text{odd-nodes-set} (\text{del-unEdge} \ v \ w \ v') \cup \{v\}$
  by (metis UnI1 odd-v)

moreover have $x = v' \Rightarrow x \in \text{odd-nodes-set} (\text{del-unEdge} \ v \ w \ v') \cup \{v\}$
  by auto

moreover have $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} \ G \cup \{v\}$ using x-prems by auto

hence $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} (\text{del-unEdge} \ v \ w \ v')$ $G$
  unfolding odd-nodes-set-def
  using $G$.degree-frame (finite (edges $G$)) by auto

hence $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} \ G \cup \{v\}$ by simp

ultimately show $x \in \text{odd-nodes-set} (\text{del-unEdge} \ v \ w \ v') \cup \{v\}$ by auto

qed

ultimately have odd-nodes-set $(\text{del-unEdge} \ v \ w \ v') \cup \{v\} = \text{odd-nodes-set} \ G$
  by auto

moreover have odd-nodes-set $G \cap \{v\} =$ \emptyset
  using parity-assms unfolding odd-nodes-set-def by auto

moreover have odd-nodes-set $(\text{del-unEdge} \ v \ w \ v') \cap \{v\} = \emptyset$
  by (metis Int-insert-left-if0 inf-bot-left inf-commute not-odd-v')

moreover have finite (odd-nodes-set $(\text{del-unEdge} \ v \ w \ v')$) $G$
  using (finite (nodes $G$)) by auto
moreover have finite (odd-nodes-set G) using (finite (nodes G)) by auto
ultimately have card(odd-nodes-set G) + card \{v\} =
    card(odd-nodes-set(del-unEdge v w v' G)) + card \{v'\}
using card-Un-disjoint[\{odd-nodes-set (del-unEdge v w v' G) \{v'\}\]card-Un-disjoint[\{odd-nodes-set G \{v\}\]
by auto
thus thesis unfolding num-of-odd-nodes-def by simp
qed

lemma del-UnEdge-odd-even:
  assumes valid-unMultigraph G finite(edges G) finite(nodes G) (v, w, v') \in edges G
  assumes parity-assms: odd (degree v G) even (degree v' G)
  shows num-of-odd-nodes (del-unEdge v w v' G) = num-of-odd-nodes G + 2
by (metis assms del-UnEdge-even-odd delete-edge-sym parity-assms valid-unMultigraph.corres)

lemma del-UnEdge-odd-odd:
  assumes valid-unMultigraph G finite(edges G) finite(nodes G) (v, w, v') \in edges G
  assumes parity-assms: odd (degree v G) odd (degree v' G)
  shows num-of-odd-nodes G = num-of-odd-nodes (del-unEdge v w v' G) + 2
proof -
  interpret G: valid-unMultigraph by fact
  have v \notin odd-nodes-set(del-unEdge v w v' G)
    by (metis G.del-UnEdge-even assms(2) assms(4) parity-assms(1))
  moreover have v' \notin odd-nodes-set(del-unEdge v w v' G)
    by (metis G.del-UnEdge-even' assms(2) assms(4) parity-assms(2))
  ultimately have vv'-disjoint: \{v, v'\} \cap odd-nodes-set(del-unEdge v w v' G) = \{
    by (metis (full-types) Int-insert-left-if0 inf-bot-left)
  moreover have extra-odd-nodes: \{v, v'\} \subseteq odd-nodes-set G
    unfolding odd-nodes-set-def
    using (v, w, v') \in edges G
    by (metis (lifting) G.E-validD empty-subset I insert-subset mem-Collect-eq parity-assms)
  moreover have odd-nodes-set G \{v, v'\} \subseteq odd-nodes-set (del-unEdge v w v' G)
proof
  fix x
  assume x-odd-set: x \in odd-nodes-set G \{v, v'\}
  hence degree x G = degree x (del-unEdge v w v' G)
    by (metis Diff-iff G.degree-frame assms(2))
  hence odd(degree x (del-unEdge v w v' G)) using x-odd-set
    unfolding odd-nodes-set-def by auto
  moreover have x \in nodes (del-unEdge v w v' G)
    using x-odd-set unfolding odd-nodes-set-def by auto
  ultimately show x \in odd-nodes-set (del-unEdge v w v' G)
    unfolding odd-nodes-set-def by auto
qed
moreover have odd-nodes-set (del-unEdge v w v' G) ⊆ odd-nodes-set G
proof
  fix x
  assume x-odd-set: x ∈ odd-nodes-set (del-unEdge v w v' G)
  hence x∉{v,v'} ⟷ odd(degree x G)
  using assms G.degree-frame unfolding odd-nodes-set-def
  by auto
  hence x∉{v,v'} ⟷ x∈odd-nodes-set G
  using x-odd-set del-UnEdge-node unfolding odd-nodes-set-def
  by auto
  moreover have x∈{v,v'} ⟷ x∈odd-nodes-set G
  using extra-odd-nodes by auto
  ultimately show x ∈ odd-nodes-set G by auto
qed
ultimately have odd-nodes-set G=odd-nodes-set (del-unEdge v w v' G) ∪ {v,v'}

by auto
thus ?thesis
proof –
  assume odd-nodes-set G=odd-nodes-set (del-unEdge v w v' G) ∪ {v,v'}
  moreover have odd-nodes-set (del-unEdge v w v' G) ∩ {v,v'} = {}
  using v*v'-disjoint by auto
  moreover have finite(odd-nodes-set (del-unEdge v w v' G))
  using assms del-UnEdge-node finite-subset unfolding odd-nodes-set-def
  by auto
  moreover have finite {v,v'} by auto
  ultimately have card(odd-nodes-set G)
  = card(odd-nodes-set (del-unEdge v w v' G)) + card{v,v'}
  unfolding num-of-odd-nodes
  by auto
  moreover have v≠v' using G.no-id ⟨(v,w,v')⟩ ∈ edges G: by auto
  hence card{v,v'}=2 by simp
  ultimately show ?thesis unfolding num-of-odd-nodes-def by simp
qed

lemma in valid-unMultigraph rem-UnPath-parity-v':
  assumes finite E is-trail v ps v'
  shows v≠v' ⟷ (odd (degree v' (rem-unPath ps G)) = even(degree v' G)) using
  assms
proof (induct ps arbitrary:v)
  case Nil
  thus ?case by (metis is-trail.simps(1) rem-unPath.simps(1))
next
  case (Cons x xs) print-cases
obtain x1 x2 x3 where x: x=(x1,x2,x3) by (metis prod-cases3)
hence rem-x:odd (degree v' (rem-unPath (x#xs) G)) = odd(degree v' (del-unEdge x1 x2 x3 (rem-unPath xs G)))

by (metis rem-unPath.simps(2) rem-unPath-com)
have $x^3 = v' \rightarrow \_thesis$
proof (cases $v = v'$)
case True
assume $x^3 = v'$
have $x^1 = v'$ using $x$ by (metis Cons.prems(2) True is-trail.simps(2))
thus $\thesis$ using $x^3 = v'$ by (metis Cons.prems(2) is-trail.simps(2) no-id $x$)
next
case False
assume $x^3 = v'$
have odd (degree $v'$ (rem-unPath ($x \neq x$) $G$)) = odd (degree $v'$ (del-unEdge $x$ $x^1 x^2 x^3$ (rem-unPath $x$ $G$))) using rem-x.
also have ... = odd (degree $v'$ (rem-unPath $x$ $G$) - 1)
proof -
have finite (edges (rem-unPath $x$ $G$))
by (metis (full-types) assms(1) finite-Diff rem-unPath-edges)
moreover have $(x^1, x^2, x^3) \in \text{edges}(\text{rem-unPath } x$ $G$)
by (metis Cons.prems(2) distinct-elim is-trail.simps(2) $x$)
moreover have $(x^3, x^2, x^1) \in \text{edges}(\text{rem-unPath } x$ $G$)
by (metis Cons.prems(2) corres distinct-elim-rev is-trail.simps(2) $x$)
ultimately show $\thESIS$
by (metis $x^3 = v'$ del-edge-undirected-degree-minus delete-edge-sym $x$)
qed
also have ... = even (degree $v'$ (rem-unPath $x$ $G$))
proof -
have $(x^1, x^2, x^3) \in E$ by (metis Cons.prems(2) is-trail.simps(2) $x$)
hence $(x^3, x^2, x^1) \in \text{edges}(\text{rem-unPath } x$ $G$)
by (metis Cons.prems(2) corres distinct-elim-rev $x$)
hence $(x^3, x^2, x^1) \in \{e \in \text{edges}(\text{rem-unPath } x$ $G$). \_fst \_e = \_v'\}$
using $x^3 = v'$ by (metis (mono-tags) _fst-conv _mem _Collect_eq)
moreover have finite $\{e \in \text{edges}(\text{rem-unPath } x$ $G$). \_fst \_e = e = \_v'\}$
using (finite $E$ by auto)
ultimately have degree $v'$ (rem-unPath $x$ $G$) ≠ 0
unfolding degree-def by auto
thus $\thESIS$ by auto
qed
also have ... = even (degree $v' G$)
using $x^3 = v'$ assms
by (metis (mono-tags) Cons.hyps Cons.prems(2) is-trail.simps(2) $x$)
finally have odd (degree $v'$ (rem-unPath ($x \neq x$) $G$)) = even (degree $v' G$).
thus $\thESIS$ by (metis False)
qed
moreover have $x^3 \neq v' \rightarrow \case$
proof (cases $v = v'$)
case True
assume $x^3 \neq v'$

have odd (degree $v'$ (rem-unPath ($x \neq x$) $G$)) = odd (degree $v'$ (del-unEdge $x$ $x^1 x^2 x^3$ (rem-unPath $x$ $G$))) using rem-x.
also have ...=odd\((\text{degree } v') (\text{rem-unPath } x s G) - 1)\)
proof
  have finite \((\text{edges } (\text{rem-unPath } x s G))\)
    by (metis (full-types) assms(1) finite-Diff rem-unPath-edges)
  moreover have \((x_1,x_2,x_3) \in \text{edges}(\text{rem-unPath } x s G)\)
    by (metis Cons.prems(2) distinct-elim is-trail.simps(2) x)
  moreover have \((x_3,x_2,x_1) \in \text{edges}(\text{rem-unPath } x s G)\)
    by (metis Cons.prems(2) corres distinct-elim-rev is-trail.simps(2) x)
  ultimately show \(?thesis\)
    using True x
  by (metis Cons.prems(2) del-edge-undirected-degree-minus is-trail.simps(2))
qed
also have ...=even\((\text{degree } v') (\text{rem-unPath } x s G)\)
proof
  have \((x_1,x_2,x_3) \in E\) by (metis Cons.prems(2) is-trail.simps(2) x)
  hence \((x_1,x_2,x_3) \in \text{edges}(\text{rem-unPath } x s G)\)
    by (metis Cons.prems(2) distinct-elim x)
  hence \((x_1,x_2,x_3) \in \{e \in \text{edges}(\text{rem-unPath } x s G). \text{fst e} = v'\}\)
    using \(v=v'\) x Cons
    by (metis (lifting, mono-tags) fst_conv is-trail.simps(2) mem-Collect-eq)
  moreover have finite \(\{e \in \text{edges}(\text{rem-unPath } x s G). \text{fst e} = v'\}\)
    using 'finite E' by auto
  ultimately have \(\text{degree } v' (\text{rem-unPath } x s G) \neq 0\)
    unfolding degree-def by auto
  thus \(?thesis\) by auto
qed
also have ...\(\neq \text{even}(\text{degree } v')\)
using \(\langle x_3 \neq v' \rangle\) assms
by (metis Cons.hyps Cons.prems(2) is-trail.simps(2) x)
finally have odd \((\text{degree } v')(\text{rem-unPath } (x \neq x s) G) \neq \text{even}(\text{degree } v' G)\).
thus \(?thesis\) by (metis True)

next
case False
assume \(x_3 \neq v'\)
  have odd \((\text{degree } v')(\text{rem-unPath } (x \neq x s) G) = \text{odd}(\text{degree } v' (\text{del-unEdge } x_1 x_2 x_3 (\text{rem-unPath } x s G)))\)
    using \(\text{rem-x} \).
  also have ...\(= \text{odd}(\text{degree } v' (\text{rem-unPath } x s G))\)
proof
  have \(v=x_1\) by (metis Cons.prems(2) is-trail.simps(2) x)
  hence \(v'\notin\{x_1,x_3\}\) by (metis (mono-tags) False \(\langle x_3 \neq v' \rangle\) empty-iff insert-iff)
  moreover have valid-unMultigraph \((\text{rem-unPath } x s G)\)
    using valid-unMultigraph-axioms by auto
  moreover have finite \((\text{edges } (\text{rem-unPath } x s G))\)
    by (metis (full-types) assms(1) finite-Diff rem-unPath-edges)
  ultimately have degree \(v'(\text{del-unEdge } x_1 x_2 x_3 (\text{rem-unPath } x s G))\)
    \(=\text{degree } v' (\text{rem-unPath } x s G)\)
    using degree-frame
  by (metis valid-unMultigraph.degree-frame)
thus \( ?\text{thesis} \) by simp 

qed 

also have \( \ldots = \text{even} \ (\deg v' G) \) 
using assms \( x \vdash x_3 \neq v' \) 
by (metis Cons.hyps Cons.prems(2) is-trail.simps(2)) 

finally have \( \text{odd} \ (\deg v' (\text{rem-unPath} \ (x \# xs) G)) = \text{even} \ (\deg v' G) \) . 
thus \( ?\text{thesis} \) by (metis False) 

qed 

ultimately show \( ?\text{case} \) by auto 

qed 

lemma (in valid-unMultigraph) rem-UnPath-parity-v: 
assumes finite \( E \) is-trail \( v \ ps \ v' \) 
shows \( v \neq v' \longleftrightarrow (\text{odd} \ (\deg v (\text{rem-unPath} \ ps G)) = \text{even} (\deg v G)) \) 
by (metis assms is-trail-rev rem-UnPath-parity-v rem-unPath-graph) 

lemma (in valid-unMultigraph) rem-UnPath-parity-others: 
assumes finite \( E \) is-trail \( v \ ps \ v' \) \( n /\in \{v, v'\} \) 
shows \( \text{even} \ (\deg n (\text{rem-unPath} \ ps G)) = \text{even} (\deg n G) \) using assms 
proof (induct \( ps \) arbitrary: \( v \)) 
  case Nil 
  thus \( ?\text{case} \) by auto 
next 
  case (Cons \( x \) \( xs \)) 
  obtain \( x_1 \ x_2 \ x_3 \) where \( x:=(x_1, x_2, x_3) \) by (metis prod-cases3) 
  hence \( \text{even} \ (\deg n (\text{rem-unPath} \ (x\#xs) G)) = \text{even} (\deg n \ (\text{del-unEdge} x_1 x_2 x_3 (\text{rem-unPath} xs G))) \) 
  by (metis rem-unPath.simps(2) rem-unPath-com) 
  have \( n=x \vdash ?\text{case} \) 
  proof 
    assume \( n=x \vdash \) 
    have \( \text{even} \ (\deg n (\text{rem-unPath} \ (x\#xs) G)) = \text{even} (\deg n \ (\text{del-unEdge} x_1 x_2 x_3 (\text{rem-unPath} xs G))) \) 
    by (metis rem-unPath.simps(2) rem-unPath-com \( x \)) 
    also have \( \ldots = \text{even}(\deg n (\text{rem-unPath} \ xs G) - 1) \) 
    proof 
      have \( \text{finite} \ (\text{edges} (\text{rem-unPath} \ xs G)) \) 
      by (metis (full-types) assms(1) finite-Diff rem-unPath-edges) 
      moreover have \( (x_1, x_2, x_3) \in \text{edges}(\text{rem-unPath} \ xs G) \) 
      by (metis Cons.prems(2) distinct-elim is-trail.simps(2) \( x \)) 
      moreover have \( (x_3, x_2, x_1) \in \text{edges}(\text{rem-unPath} \ xs G) \) 
      by (metis Cons.prems(2) corres distinct-elim-rev is-trail.simps(2) \( x \)) 
      ultimately show \( ?\text{thesis} \) 
      using \( n = x \vdash \text{del-edge-undirected-degree-minus} \) 
      by auto 
    qed 
  also have \( \ldots = \text{odd}(\deg n (\text{rem-unPath} \ xs G)) \) 
  proof 
    have \( (x_1, x_2, x_3) \in E \) by (metis Cons.prems(2) is-trail.simps(2) \( x \))
hence \((x_3, x_2, x_1) \in \text{edges} (\text{rem-unPath } xs \ G)\)
by \((\text{metis } \text{Cons.prems}(2) \ \text{corres distinct-elim-rev } x)\)
hence \((x_3, x_2, x_1) \in \{ e \in \text{edges} (\text{rem-unPath } xs \ G). \ \text{fst } e = n \}\)
using \((n=x3)\) by \((\text{metis } \text{mono-tags} \ \text{fst-cone mem-Collect-eq})\)
moreover have \(\text{finite} \{ e \in \text{edges} (\text{rem-unPath } xs \ G). \ \text{fst } e = n \}\)
using \(\text{finite } E\) by auto
ultimately have \(\text{degree } n (\text{rem-unPath } xs \ G) \neq 0\)
unfolding \(\text{degree-def}\) by auto
thus \(\text{thesis}\) by auto
qed
also have ...=\(\text{even}(\text{degree } n \ G)\)
proof –
have \(x_3 \neq v\) by \((\text{metis } \text{assms}(3) \ \text{insert-iff})\)
hence \(\text{odd} (\text{degree } x_3 (\text{rem-unPath } xs \ G)) = \text{even}(\text{degree } x_3 \ G)\)
using \(\text{Cons } \text{assms}\)
by \((\text{metis } \text{is-trail}.\text{simps}(2) \ \text{rem-UnPath-parity-v } x)\)
thus \(\text{thesis}\) using \((n=x3)\) by auto
qed
finally have \(\text{even} (\text{degree } n (\text{rem-unPath } (x\#xs) \ G)) = \text{even}(\text{degree } n \ G)\).
thus \(\text{thesis}\).
qed
moreover have \(n \neq x_3 \implies \text{?case}\)
proof –
assume \(n \neq x_3\)
have \(\text{even} (\text{degree } n (\text{rem-unPath } (x\#xs) \ G)) = \text{even} (\text{degree } n (\ \text{del-unEdge } x_1 x_2 x_3 \ (\text{rem-unPath } xs \ G))))\)
by \((\text{metis } \text{rem-unPath}.\text{simps}(2) \ \text{rem-unPath-com } x)\)
also have ...=\(\text{even}(\text{degree } n \ (\text{rem-unPath } xs \ G))\)
proof –
have \(v=x_1\) by \((\text{metis } \text{Cons.prems}(2) \ \text{is-trail}.\text{simps}(2) \ x)\)
hence \(n \notin \{x_1, x_3\}\) by \((\text{metis } \text{Cons.prems}(3) \ (n \neq x_3). \ \text{insertE insertI1 singletonE})\)
moreover have \(\text{valid-Multigraph} (\text{rem-unPath } xs \ G)\)
using \(\text{valid-Multigraph-axioms}\) by auto
moreover have \(\text{finite} (\text{edges} (\text{rem-unPath } xs \ G))\)
by \((\text{metis } \text{full-types} \ \text{assms}(1) \ \text{finite-Diff rem-unPath-edges})\)
ultimately have \(\text{degree } n (\text{del-unEdge } x_1 x_2 x_3 \ (\text{rem-unPath } xs \ G)) = \text{degree } n \ (\text{rem-unPath } xs \ G)\) using \(\text{degree-frame}\)
by \((\text{metis } \text{valid-Multigraph}.\text{degree-frame})\)
thus \(\text{thesis}\) by simp
qed
also have ...=\(\text{even}(\text{degree } n \ G)\)
using \(\text{Cons } \text{assms}(n \neq x_3)\) \(x\) by auto
finally have \(\text{even} (\text{degree } n (\text{rem-unPath } (x\#xs) \ G)) = \text{even}(\text{degree } n \ G)\).
thus \(\text{thesis}\).
qed
ultimately show \(\text{?case}\) by auto
qed
lemma (in valid-unMultigraph) rem-UnPath-even:
  assumes finite E finite V is-trail v ps v'
  assumes parity-assms: even (degree v' G)
  shows num-of-odd-nodes (rem-unPath ps G) = num-of-odd-nodes G 
            + (if even (degree v G) \wedge v \neq v' then 2 else 0) using assms
proof (induct ps arbitrary: v)
  case Nil
  thus ?case by auto
next
  case (Cons x xs)
  obtain x1 x2 x3 where x:x=(x1,x2,x3) by (metis prod-cases3)
  have fin-nodes: finite (nodes (rem-unPath xs G)) using Cons by auto
  have fin-edges: finite (edges (rem-unPath xs G)) using Cons by auto
  have valid-rem-xs: valid-unMultigraph (rem-unPath xs G) using valid-unMultigraph-axioms
    by auto
  have x-in: (x1,x2,x3) \in edges (rem-unPath xs G)
    by (metis (full-types) Cons.prems (2) distinct-elim is-trail.simps(2) x)
  have even (degree x1 (rem-unPath xs G)) \implies even (degree x3 (rem-unPath xs G)) \implies ?case
    proof -
      assume parity-x1-x3: even (degree x1 (rem-unPath xs G))
      even (degree x3 (rem-unPath xs G))
      have num-of-odd-nodes (rem-unPath (x#xs) G) = num-of-odd-nodes
        (del-unEdge x1 x2 x3 (rem-unPath xs G))
        by (metis rem-unPath.simps(2) rem-unPath-com x)
      also have ... = num-of-odd-nodes (rem-unPath xs G)+2
        using parity-x1-x3 fin-nodes fin-edges valid-rem-xs x-in del-UnEdge-even-even
        by metis
      also have ... = num-of-odd-nodes G + (if even (degree x3 G) \wedge x3 \neq v' then 2 else 0) + 2
        using Cons.hyps[OF \{finite E \{finite V\}, of x3\} \is-trail v (x \# xs) v'
         \(even (degree v' G)\) \ x]
        by auto
      also have ... = num-of-odd-nodes G+2
        proof -
          have even (degree x3 G) \wedge x3 \neq v' \longleftrightarrow odd (degree x3 (rem-unPath xs G))
            using Cons.prems assms
            by (metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x)
          thus ?thesis using parity-x1-x3(2) by auto
          qed
        also have ... = num-of-odd-nodes G + (if even (degree v G) \wedge v \neq v' then 2 else 0)
        proof -
          have x1 \neq x3 by (metis valid-rem-xs valid-unMultigraph.no-id x-in)
          moreover hence x1 \neq v'
            using Cons assms
            by (metis is-trail.simps(2) parity-x1-x3(1) rem-UnPath-parity-v' x)
          qed
  qed
ultimately have $x \not\in \{x, v'\}$ by auto
hence even $(\text{degree } x_1 G)$
using Cons.prems(3) assms(1) assms(2) parity-x1-x3(1)
by (metis (full-types) is-trail.simps(2) rem-UnPath-parity-others x)
hence even $(\text{degree } x_1 G) \land x_1 \not\neq v'$ by auto
hence even $(\text{degree } v G) \land v \not\neq v'$ by (metis Cons.prems(3) is-trail.simps(2))

$\therefore$ thus \textit{thesis} by auto

\textit{qed}

finally have \textit{num-of-odd-nodes} $(\text{rem-unPath } (x \# x s) G) = \textit{num-of-odd-nodes} G + (\text{if } \text{even}(\text{degree } v G) \land v \not\neq v' \text{ then } 2 \text{ else } 0)$.
\textit{thesis} by auto
\textit{qed}

\textit{moreover have } even $(\text{degree } x_1 (\text{rem-unPath } x s G)) \iff \textit{odd}(\text{degree } x_3 (\text{rem-unPath } x s G)) \iff \textit{thesis}$
proof

assume parity-x1-x3: even $(\text{degree } x_1 (\text{rem-unPath } x s G))$
odd $(\text{degree } x_3 (\text{rem-unPath } x s G))$

have \textit{num-of-odd-nodes} $(\text{rem-unPath } (x \# x s) G) = \textit{num-of-odd-nodes} (\text{del-unEdge } x_1 x_2 x_3 (\text{rem-unPath } x s G))$
by (metis rem-unPath.simps(2) rem-unPath-com x)
also have ... \textit{=num-of-odd-nodes} $(\text{rem-unPath } x s G)$
using parity-x1-x3 fin-nodes fin-edges valid-rem-xs x-in
by (metis del-UnEdge-even-odd)
also have ... \textit{=num-of-odd-nodes} G \(+ (\text{if } \text{even}(\text{degree } x_3 G) \land x_3 \not\neq v' \text{ then } 2 \text{ else } 0)\)

using Cons.hyps Cons.prems(3) assms(1) assms(2) parity-assms x
by auto
also have ... \textit{=num-of-odd-nodes} G \(+ 2\)
proof

have even $(\text{degree } x_3 G) \land x_3 \not\neq v' \iff \textit{odd}(\text{degree } x_3 (\text{rem-unPath } x s G))$
using Cons.prems assms
by (metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x)
thus \textit{thesis} using parity-x1-x3(2) by auto
\textit{qed}

also have ... \textit{=num-of-odd-nodes} G \(+ (\text{if } \text{even}(\text{degree } v G) \land v \not\neq v' \text{ then } 2 \text{ else } 0)\)

proof

have $x_1 \not\neq x_3$ by (metis valid-rem-xs valid-unMultigraph.no-id x-in)

moreover hence $x_1 \not\neq v'$
using Cons assms
by (metis is-trail.simps(2) parity-x1-x3(1) rem-UnPath-parity-v' x)
ultimately have $x \not\in \{x, v'\}$ by auto
hence even $(\text{degree } x_1 G)$
using Cons.prems(3) assms(1) assms(2) parity-x1-x3(1)
by (metis (full-types) is-trail.simps(2) rem-UnPath-parity-others x)
hence even $(\text{degree } x_1 G) \land x_1 \not\neq v'$ using $x_1 \not\neq v'$ by auto
hence even $(\text{degree } v G) \land v \not\neq v'$ by (metis Cons.prems(3) is-trail.simps(2))
thus \( \text{thesis} \) by auto

qed

finally have num-of-odd-nodes \((\text{rem-unPath} (x \# \text{xs}) G)\) =
num-of-odd-nodes \(G + (\text{if even(degree} v G) \land v \neq v' \text{ then 2 else 0})\).

thus \( \text{thesis} \).

qed

moreover have odd \((\text{degree} x_1 (\text{rem-unPath} \text{xs} G))\) \(\implies\)
even \((\text{degree} x_3 (\text{rem-unPath} \text{xs} G))\) \(\implies\) \(?\text{case}\)

case True
have \(x_1 \neq x_3\) by (metis valid-rem-xs valid-unMultigraph.no-id x-in)

moreover have is-trail \(x_3 \text{xs} v'\)

ultimately have odd \((\text{degree} x_1 (\text{rem-unPath} \text{xs} G))\)

\(\iff\) odd\((\text{degree} x_1 G)\)

using True parity-x1-x3(1) \text{rem-UnPath-parity-others} x Cons.prems(3)
assms(1) assms(2)

by auto

hence odd\((\text{degree} x_1 G)\) by (metis parity-x1-x3(1))

thus \( \text{thesis} \)

by (metis \(\text{mono-tags} \) Cons.prems(3) Nat.add-0-right is-trail.simps(2))

next

case False
then show \( \text{thesis} \) by auto
finally have num-of-odd-nodes (rem-unPath (x ≠ xs) G) =
num-of-odd-nodes G + (if even (degree v G) ∧ v ≠ v’ then 2 else 0).

thus thesis.

qed

moreover have odd (degree x1 (rem-unPath xs G)) →
odd (degree x3 (rem-unPath xs G)) → ?case

proof –
assume parity-x1-x3: odd (degree x1 (rem-unPath xs G))
odd (degree x3 (rem-unPath xs G))
have num-of-odd-nodes (rem-unPath (x ≠ xs) G) = num-of-odd-nodes
(del-unEdge x1 x2 x3 (rem-unPath xs G))
by (metis rem-unPath.simps(2) rem-unPath-com x)
also have ... = num-of-odd-nodes (rem-unPath xs G) − (2 :: nat)
using del-UnEdge-odd-odd
by (metis add-implies-diff fin-edges fin-nodes parity-x1-x3 valid-rem-xs x-in)

also have ...
num-of-odd-nodes G + (if even (degree x3 G) ∧ x3 ≠ v’ then 2 else 0) − (2 :: nat)

using Cons assms
by (metis is-trail.simps(2) x)
also have ...
num-of-odd-nodes G

proof –
have even (degree x3 G) ∧ x3 ≠ v’ ↔ odd (degree x3 (rem-unPath xs G))
using Cons.prems assms
by (metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x)
thus thesis using parity-x1-x3(2) by auto

qed
also have ...
num-of-odd-nodes G + (if even (degree v G) ∧ v ≠ v’ then 2 else 0)

proof (cases v ≠ v’)
case True
have x1 ≠ x3 by (metis valid-rem-xs valid-unMultigraph.no-id x-in)
moreover have is-trail x3 xs v’
by (metis Cons.prems(3) is-trail.simps(2) x)
ultimately have odd (degree x1 (rem-unPath xs G))
←→ odd (degree x1 G)
using True Cons.prems(3) assms(1) assms(2) parity-x1-x3(1) rem-UnPath-parity-others

by auto
hence odd (degree x1 G) by (metis parity-x1-x3(1))
thus thesis
by (metis (mono-tags) Cons.prems(3) Nat.add-0-right is-trail.simps(2))

next
case False
thus thesis by (metis (mono-tags) add-0-iff)

qed
finally have num-of-odd-nodes (rem-unPath (x#xs) G) =
  num-of-odd-nodes G + (if even (degree v G) ∧ v ≠ v' then 2 else
0).
  thus ?thesis .
qed
ultimately show ?case by metis
qed

lemma (in valid-unMultigraph) rem-UnPath-odd:
  assumes finite E finite V is-trail v ps v'
  assumes parity-assms: odd (degree v' G)
  shows num-of-odd-nodes (rem-unPath ps G) = num-of-odd-nodes G
  + (if odd (degree v G) ∧ v ≠ v' then −2 else 0) using assms
proof (induct ps arbitrary: v)
  case Nil
  thus ?case by auto
next
  case (Cons x xs)
  obtain x1 x2 x3 where x := (x1, x2, x3) by (metis prod-cases3)
  have fin-nodes: finite (nodes (rem-unPath xs G)) using Cons by auto
  have fin-edges: finite (edges (rem-unPath xs G)) using Cons by auto
  have valid-rem-xs: valid-unMultigraph (rem-unPath xs G) using valid-unMultigraph-axioms
    by auto
  have x-in: (x1, x2, x3) ∈ edges (rem-unPath xs G)
    by (metis (full-types) Cons.prems(3) distinct-elim is-trail.simps(2) x)
  have even (degree x1 (rem-unPath xs G))
    ⇒ even (degree x3 (rem-unPath xs G)) ⇒ ?case
  proof –
    assume parity-x1-x3: even (degree x1 (rem-unPath xs G))
    have odd (degree x3 (rem-unPath xs G))
      using Cons.prems assms by (metis is-trail.simps(2) parity-x1-x3 rem-UnPath-parity-v x)
  also have ... = num-of-odd-nodes (rem-unPath xs G) + 2
  using parity-x1-x3 fin-nodes fin-edges valid-rem-xs x-in del-UnEdge-even-even
    by auto
  else 0 \)+2
  using Cons.hyps[OF \{finite E \\{finite V, of x3\} \is-trail v (x ≠ xs) v'\}
      \{odd (degree v' G)\} x]
  by auto
  also have ... = num-of-odd-nodes G + (if odd (degree x3 G) ∧ x3 ≠ v' then −2 else 0)
  using Cons.prems assms
  proof –
    have odd (degree x3 G) ∧ x3 ≠ v' \←→\ even (degree x3 (rem-unPath xs G))
      using Cons.prems assms
    by (metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x)
thus \(?thesis\) using \texttt{parity-x1-x3(2)} by \texttt{auto}

qed

also have \(\ldots=\text{num-of-odd-nodes } G+ (\text{if odd}(\text{degree } v G) \land v \neq v' \text{ then } -2 \text{ else 0)}\)

proof (cases \(v \neq v'\))

\begin{itemize}
  \item \texttt{case True}
    \begin{itemize}
      \item \texttt{have } \(x1 \neq x3\) by \texttt{(metis valid-rem-xs valid-unMultigraph.no-id x-in)}
      \item \texttt{moreover have } \texttt{is-trail } \(x3\) \(\!\vdash\! x'\)
        by \\texttt{(metis \textit{Cons.prems(3)} is-trail.simps(2) x)}
      \item \texttt{ultimately have } \texttt{even } \(\langle\text{degree } x1 \ (\text{rem-unPath } x\!\vdash\! x G)\rangle \ \iff \ \text{even } \langle\text{degree } x1 \ G\rangle\)
        using \texttt{True Cons.prems(3) assms(1) assms(2) parity-x1-x3(1)}
        \texttt{rem-UnPath-parity-others x}
        by \texttt{auto}
      \item \texttt{hence } \texttt{even } \(\langle\text{degree } x1 \ G\rangle\) \texttt{by } \texttt{(metis parity-x1-x3(1))}
      \item \texttt{thus } \texttt{?thesis by } \texttt{(metis hide-lams, mono-tags Cons.prems(3) is-trail.simps(2)
        monoid-add-class.add.right-neutral x)}
    \end{itemize}
  \item \texttt{next}
    \begin{itemize}
      \item \texttt{case False}
      \item \texttt{then show } \texttt{?thesis by } \texttt{auto}
    \end{itemize}
\end{itemize}

qed

finally have \(\text{num-of-odd-nodes } (\text{rem-unPath } (x\#xs) \!\vdash\! G) = \text{num-of-odd-nodes } G + (\text{if odd}(\text{degree } v G) \land v \neq v' \text{ then } -2 \text{ else 0)}\)

thus \texttt{?thesis}.

qed

moreover have \(\text{even } (\text{degree } x1 \ (\text{rem-unPath } x\!\vdash\! x G)) \implies \text{odd } (\text{degree } x3 \ (\text{rem-unPath } x\!\vdash\! x G)) \implies \text{?case}\)

proof

assume \texttt{parity-x1-x3: even } \(\langle\text{degree } x1 \ (\text{rem-unPath } x\!\vdash\! x G)\rangle\)
\(\implies \text{odd } (\text{degree } x3 \ (\text{rem-unPath } x\!\vdash\! x G))\)

have \(\text{num-of-odd-nodes } (\text{rem-unPath } (x\#xs) \!\vdash\! G) = \text{num-of-odd-nodes}\)
\(\langle\text{del-UnEdge } x1 \times2 \times3 \ (\text{rem-unPath } x\!\vdash\! x G)\rangle\)
\(\text{by } \texttt{(metis rem-unPath.simps(2) rem-unPath-com x)}\)

also have \(\ldots=\text{num-of-odd-nodes } G + (\text{if odd}(\text{degree } x3 G) \land x3 \neq v' \text{ then } -2 \text{ else 0)}\)

using \texttt{Cons.hyps[OF 'finite E' 'finite V', of x3] Cons.prems(3) assms(1) assms(2) parity-assms x}
\texttt{by } \texttt{auto}

also have \(\ldots=\text{num-of-odd-nodes } G\)

proof

\(\texttt{have odd}(\text{degree } x3 G) \land x3 \neq v' \iff \text{even } (\text{degree } x3 \ (\text{rem-unPath } x\!\vdash\! x G))\)

using \texttt{Cons.prems assms}
\texttt{by } \texttt{(metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x)}

26
thus \( ?\text{thesis} \) using \text{parity-x1-x3}(2) by auto

qed

also have \( \ldots = \text{num-of-odd-nodes} \ G + (\text{if odd}(\text{degree} \ v \ G) \land v \neq v' \text{ then } -2 \text{ else } 0) \)

proof (cases \( v \neq v' \))

\begin{itemize}
  \item case True
    \begin{itemize}
      \item have \( x1 \neq x3 \) by \text{metis valid-rem-xs valid-unMultigraph.no-id x-in}
      \item moreover have \( \text{is-trail} \ x3 \ x5 \ v' \)
        \begin{itemize}
          \item by \text{metis Cons.prems(3) is-trail.simps(2) x}
        \end{itemize}
      \item ultimately have \( \text{even} \ (\text{degree} \ x1 \ (\text{rem-unPath} \ x \ G)) \)
        \begin{itemize}
          \item \text{using True Cons.prems(3) assms(1) assms(2) parity-x1-x3(1)}
        \end{itemize}
    \end{itemize}
  \item case False
    \begin{itemize}
      \item then show \( ?\text{thesis} \) by auto
    \end{itemize}
\end{itemize}

qed

finally have \( \text{num-of-odd-nodes} \ (\text{rem-unPath} \ (x \# x) \ G) = \text{num-of-odd-nodes} \ G + (\text{if odd}(\text{degree} \ v \ G) \land v \neq v' \text{ then } -2 \text{ else } 0) \)

thus \( ?\text{thesis} \).

qed

moreover have \( \text{odd} \ (\text{degree} \ x1 \ (\text{rem-unPath} \ x \ G)) \implies \text{even}(\text{degree} \ x3 \ (\text{rem-unPath} \ x \ G)) \implies ?\text{case} \)

proof

\begin{itemize}
  \item assume \( \text{parity-x1-x3} : \text{odd} \ (\text{degree} \ x1 \ (\text{rem-unPath} \ x \ G)) \)
    \begin{itemize}
      \item even \( (\text{degree} \ x3 \ (\text{rem-unPath} \ x \ G)) \)
    \end{itemize}
  \item have \( \text{num-of-odd-nodes} \ (\text{rem-unPath} \ (x \# x) \ G) = \text{num-of-odd-nodes} \)
    \begin{itemize}
      \item \text{(del-unEdge} \ x1 \ x2 \ x3 \ (\text{rem-unPath} \ x \ G)) \)
    \end{itemize}
  \item by \text{metis \text{rem-unPath}.simp(2) rem-unPath-com x)
  \item also have \( \ldots = \text{num-of-odd-nodes} \ (\text{rem-unPath} \ x \ G) \)
    \begin{itemize}
      \item using \( \text{parity-x1-x3} \) \text{fin-nodes fin-edges valid-rem-x s x-in}
    \end{itemize}
  \item by \text{metis \text{del-UnEdge-odd-even}}
  \item also have \( \ldots = \text{num-of-odd-nodes} \ G + (\text{if odd}(\text{degree} \ x3 \ G) \land x3 \neq v' \text{ then } -2 \text{ else } 0) \)
    \begin{itemize}
      \item \text{using Cons.hyps Cons.prems(3) assms(1) assms(2) parity-assms x}
    \end{itemize}
  \item by auto
  \item also have \( \ldots = \text{num-of-odd-nodes} \ G + (- 2) \)
    \begin{itemize}
      \item \text{proof}
        \begin{itemize}
          \item have \( \text{odd}(\text{degree} \ x3 \ G) \land x3 \neq v' \implies \text{even}(\text{degree} \ x3 \ (\text{rem-unPath} \ x \ G)) \)
            \begin{itemize}
              \item using Cons.prems assms
            \end{itemize}
          \item by \text{metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x}
        \end{itemize}
    \end{itemize}
  \item hence \( \text{odd}(\text{degree} \ x3 \ G) \land x3 \neq v' \text{ by } \text{metis parity-x1-x3(2)} \)
    \begin{itemize}
      \item thus \( \text{thesis} \) by auto
    \end{itemize}
  \item qed
  \item also have \( \ldots = \text{num-of-odd-nodes} \ G + (\text{if odd}(\text{degree} \ v \ G) \land v \neq v' \text{ then } -2 \text{ else }\}

27
proof

have \( x_1 \neq x_3 \) by (metis valid-rem-xs valid-unMultigraph.no-id x-in)
moreover hence \( x_1 \neq v' \)
using Cons assms
by (metis is-trail.simps(2) parity-x1-x3(1) rem-UnPath-parity-v' x)
ultimately have \( x_1 \notin \{x_3, v'\} \) by auto
hence \( \text{odd}(\text{degree } x_1 G) \)
using Cons.prems(3) assms(1) assms(2) parity-x1-x3(1)
by (metis (full-types) is-trail.simps(2) rem-UnPath-parity-others x)
thus \( ?\text{thesis} \) by auto
qed

also have \( \text{num-of-odd-nodes } (\text{rem-unPath } x#xs G) = \text{num-of-odd-nodes } G + (\text{if } \text{odd}(\text{degree } v G) \land v \neq v' \text{ then } -2 \text{ else } 0) \)

thus \( ?\text{thesis} \).
qed

moreover have \( \text{odd}(\text{degree } x_1 \ (\text{rem-unPath } xs G)) \implies \text{odd}(\text{degree } x_3 \ (\text{rem-unPath } xs G)) \implies ?\text{case} \)
proof
assume parity-x1-x3: \( \text{odd}(\text{degree } x_1 \ (\text{rem-unPath } xs G)) \)
\( \text{odd}(\text{degree } x_3 \ (\text{rem-unPath } xs G)) \)

have \( \text{num-of-odd-nodes } (\text{rem-unPath } x#xs G) = \text{num-of-odd-nodes } (\text{del-unEdge } x_1 x_2 x_3 \ (\text{rem-unPath } xs G)) \)
by (metis rem-unPath.simps(2) rem-unPath-com x)
also have \( \ldots = \text{num-of-odd-nodes } (\text{rem-unPath } xs G) - (2::nat) \)
using del-UnEdge-odd-odd
by (metis add-implies-diff fin-edges fin-nodes parity-x1-x3 valid-rem-xs x-in)

also have \( \ldots = \text{num-of-odd-nodes } G - (2::nat) \)
proof
have \( \text{odd}(\text{degree } x_3 G) \land x_3 \neq v' \iff \text{even}(\text{degree } x_3 \ (\text{rem-unPath } xs G)) \)
using Cons.prems assms
by (metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x)

\( \neg (\text{odd}(\text{degree } x_3 G) \land x_3 \neq v') \) by (metis parity-x1-x3(2))

have \( \text{num-of-odd-nodes } (\text{rem-unPath } xs G) = \text{num-of-odd-nodes } G + (\text{if } \text{odd}(\text{degree } x_3 G) \land x_3 \neq v' \text{ then } -2 \text{ else } 0) \)
by (metis Cons.hyps Cons.prems(3) assms(1) assms(2) is-trail.simps(2) parity-assms x)
thus \( ?\text{thesis} \)
using \( \neg (\text{odd}(\text{degree } x_3 G) \land x_3 \neq v') \) by auto
qed

also have \( \ldots = \text{num-of-odd-nodes } G + (\text{if } \text{odd}(\text{degree } v G) \land v \neq v' \text{ then } -2 \text{ else } 0) \)

proof
have \( x_1 \neq x_3 \) by (metis valid-rem-xs valid-unMultigraph.no-id x-in)
moreover hence \( x_1 \neq v' \)
using Cons assms
by (metis is-trail.simps(2) parity-\( x_1 \)-\( x_3 \)(1) rem-UnPath-parity-v' \( x \))
ultimately have \( x_1 \notin \{x_3, v'\} \) by auto
hence odd(degree \( x_1 \) G)
using Cons.prems(3) assms(1) assms(2) parity-\( x_1 \)-\( x_3 \)(1)
by (metis (full-types) is-trail.simps(2) rem-UnPath-parity-others x)
hence odd(degree \( x_1 \) G) \&\& \( x_1 \neq v' \) using \( x_1 \neq v' \)
hence odd(degree v G) \&\& v \neq v' using \( \langle x_1 \neq v' \rangle \) by auto

hence \( x \in \text{odd-nodes-set} \ G \)
using Cons.prems(3) E-validD(1) x unfolding odd-nodes-set-def
by auto
moreover have \( v' \in \text{odd-nodes-set} \ G \)
using is-path-memb[OF is-trail-intro[OF assms(3)]] parity-assms
unfolding odd-nodes-set-def
by auto
ultimately have \( \{v, v'\} \subseteq \text{odd-nodes-set} \ G \) by auto
moreover have \( v \neq v' \) by (metis \( \langle \text{odd (degree v G) \&\& v \neq v'} \rangle \))
hence \( \text{card}(v, v') = 2 \) by auto
moreover have finite(odd-nodes-set G)
using finite V unfolding odd-nodes-set-def
by auto
ultimately have \( \text{num-of-odd-nodes} \ G \geq 2 \) by (metis card-mono num-of-odd-nodes-def)

thus ?thesis using \( \langle \text{odd (degree v G) \&\& v \neq v'} \rangle \) by auto
qed
finally have \( \text{num-of-odd-nodes} \ (\text{rem-unPath} \ (x\#xs) \ G) = \text{num-of-odd-nodes} \ G + (\text{if odd(dgree v G) \&\& v \neq v' then} -2 \text{ else } 0) \) .
thus ?thesis .
qed
ultimately show ?case by metis
qed
3 Connectivity

definition (in valid-unMultigraph) connected::bool where
  connected ≡ ∀ v∈V. ∀ v′∈V. v≠v′ → (∃ ps. is-path v ps v′)

lemma (in valid-unMultigraph) connected ⇒ ∀ v∈V. ∀ v′∈V. v≠v′→(∃ ps. is-trail v ps v′)
proof (rule,rule,rule)
  fix v v'
  assume v∈V v′∈V v≠v'
  assume connected
  obtain ps where is-path v ps v′ by (metis connected ⟨v∈V⟩ ⟨v′∈V⟩ ⟨v≠v′⟩ connected-def)
  then obtain ps' where is-trail v ps' v'
  proof (induct ps arbitrary:v )
    case Nil
    thus ?case by (metis is-trail.simps(1) is-path.simps(1))
  next
    case (Cons x xs)
    obtain x1 x2 x3 where x:x=(x1,x2,x3) by (metis prod-cases3)
    have is-path x3 xs v' by (metis Cons.prems(2) is-path.simps(2) x)
    moreover have ∨ps'. is-trail x3 ps' v'⇒ thesis
    proof –
      fix ps'
      assume is-trail x3 ps' v'
      hence (x1,x2,x3)∈set ps' ∧ (x3,x2,x1)∈set ps' → is-trail v (x#ps') v'
      by (metis Cons.prems(2) is-trail.simps(2) is-path.simps(2) x)
      moreover have (x1,x2,x3)∈set ps' ⇒ ∃ ps1. is-trail v ps1 v'
      proof –
        assume (x1,x2,x3)∈set ps'
        then obtain ps1 ps2 where ps'=ps1@(x1,x2,x3)#ps2 by (metis split-list)
        hence is-trail v (x#ps2) v'
        using is-trail x3 ps' v' x
        by (metis Cons.prems(2) is-trail.simps(2)
            is-trail-split is-path.simps(2))
        thus ?thesis by rule
      qed
    moreover have (x3,x2,x1)∈set ps' ⇒ ∃ ps1. is-trail v ps1 v'
    proof –
      assume (x3,x2,x1)∈set ps'
      then obtain ps1 ps2 where ps'=ps1@(x3,x2,x1)#ps2 by (metis split-list)
      hence is-trail v ps2 v'
      using is-trail x3 ps' v' x
      by (metis Cons.prems(2) is-trail.simps(2)
            is-trail-split is-path.simps(2))
      thus ?thesis by rule
    qed
ultimately show thesis using Cons by auto qed
ultimately show ?case using Cons by auto qed
thus \( \exists \, \text{ps}. \, \text{is-trail v ps v'} \) by rule qed

lemma (in valid-unMultigraph) no-rep-length: is-trail v ps v' \( \Rightarrow \) length ps = card(set ps)
by (induct ps arbitrary: v, auto)

lemma (in valid-unMultigraph) path-in-edges: is-trail v ps v' \( \Rightarrow \) set ps \( \subseteq \) E
proof (induct ps arbitrary: v)
  case Nil
  show ?case by auto
next
case (Cons x xs)
  obtain x1 x2 x3 where x = (x1, x2, x3) by (metis prod-cases3)
  hence is-trail x3 xs v' using Cons by auto
  hence set xs \( \subseteq \) E using Cons by auto
  moreover have x \( \in \) E using Cons by (metis is-trail-intro is-path.simps(2) x)
  ultimately show ?case by auto qed

lemma (in valid-unMultigraph) trail-bound:
assumes finite E is-trail v ps v'
shows length ps \( \leq \) card E
by (metis (hide-lams, no-types) assms(1) assms(2) card-mono no-rep-length path-in-edges)

definition (in valid-unMultigraph) exist-path-length :: 'v \Rightarrow nat \Rightarrow bool where
exist-path-length v l \( \equiv \exists \, \text{v' ps}. \, \text{is-trail v'} \, \text{ps v} \land \text{length ps} = \text{l}

lemma (in valid-unMultigraph) longest-path:
assumes finite E n \( \in \) V
shows \( \exists \, \text{v. max-path: is-trail v max-path n} \land \) (\( \forall \, \text{v'}. \forall \, \text{e \in E. \, \neg is-trail v' (e\#max-path n)} \))
proof (rule ccontr)
  assume contro: \( \exists \, \text{v max-path: is-trail v max-path n} \land \) (\( \forall \, \text{v'}. \forall \, \text{e \in E. \, \neg is-trail v' (e\#max-path n)} \))
  hence induct:(\( \forall \, \text{v max-path: is-trail v max-path n} \land \) (\( \forall \, \text{v'}. \forall \, \text{e \in E. \, \neg is-trail v' (e\#max-path n)} \))) by auto
  have is-trail n \[ n \] using \( \text{n \in V} \) by auto
  hence exist-path-length n 0 unfolding exist-path-length-def by auto
  moreover have \( \forall \, \text{y. exist-path-length n y} \Rightarrow y \leq \text{card E} \)
    using trail-bound[OF finite E] unfolding exist-path-length-def by auto
  hence bound:\( \forall \, \text{y. exist-path-length n y} \Rightarrow y < \text{card E + 1} \) by auto
  ultimately have exist-path-length n (GREATEST x. exist-path-length n x) using
GreatestI by auto
then obtain $v$ max-path where
max-path:is-trail $v$ max-path $n$ length max-path=$(\text{GREATEST } x. \text{exist-path-length } n \ x)$
by (metis exist-path-length-def)
hence $\exists v'. \text{is-trail } v' (e\#\text{max-path}) \ n$ using induct by metis
hence exist-path-length $n$ (length max-path +1)
by (metis One-nat-def exist-path-length-def list.size(4))
hence length max-path +1 $\leq$ (GREATEST $x$. exist-path-length $n$ $x$) by (metis Greatest-le bound)
hence length max-path +1 $\leq$ length max-path using max-path by auto
thus False by auto
qed

lemma even-card':
assumes even(card $A$ ) $x\in A$
shows $\exists y \in A. y\neq x$
proof (rule ccontr)
assume $\neg (\exists y \in A. y \neq x)$
hence $\forall y \in A. y = x$ by auto
hence $A = \{x\}$ by (metis all-not-in-conv assms(2) insertI2 mk-disjoint-insert)
hence card($A$) =1 by auto
thus False using (even(card $A$ )) by auto
qed

lemma odd-card:
assumes finite $A$ odd(card $A$)
shows $\exists x. x \in A$
by (metis all-not-in-conv assms(2) card-empty even-zero)

lemma (in valid-unMultigraph) extend-distinct-path:
assumes finite $E$ is-trail $v'$ $ps$ $v$
assumes parity-assms: (even (degree $v'$ $G$)$\land v'\neq v) \lor (\text{odd (degree } v' G) \land v'=v)$
shows $\exists e \ v1. \text{is-trail } v1 (e\#ps) \ v$
proof –
have (even (degree $v'$ $G$)$\land v'\neq v) \implies \text{odd (degree } v' (\text{rem-unPath } ps \ G))$
by (metis assms(1) assms(2) rem-UnPath-parity-v)
moreover have (odd (degree $v'$ $G$)$\land v'=v) \implies \text{odd (degree } v' (\text{rem-unPath } ps \ G))$
by (metis assms(1) assms(2) rem-UnPath-parity-v)
ultimately have odd(degree $v'$ (rem-unPath $ps$ $G$)) using parity-assms by auto
hence odd (card $\{e. \text{fst } e\ = v' \land e\in \text{edges } G \ - \ (\text{set } ps \cup \text{set (rev-path } ps))\}$)
using rem-unPath-edges unfolding degree-def
by (metis (lifting, no-types) Collect-cong)
hence $\{e. \text{fst } e\ = v' \land e\in E \ - \ (\text{set } ps \cup \text{set (rev-path } ps))\} \neq \{\}$
by (metis empty_iff finite.emptyI odd-card)
then obtain $v0 \ w$ where $v0w$: $(v',w,v0)\in E$ $(v',w,v0)\notin \text{set } ps \cup \text{set (rev-path } ps)$ by auto
hence is-trail $v0 ((v0, w, v') \# ps) v$
by (metis (hide-lams, mono-tags) Un-iff assms(2) corros in-set-rev-path is-trail.simps(2))

thus ?thesis by metis

qed

replace an edge (or its reverse in a path) by another path (in an undirected graph)

fun replace-by-UnPath :: 
'v, w, v' path ⇒ 
'v × 'w × 'v ⇒ 
'v, v' path
where
replace-by-UnPath [] - - = [] |
replace-by-UnPath (x#xs) (v,e,v') ps =
(if x=(v,e,v') then ps@replace-by-UnPath xs (v,e,v') ps
else if x=(v',e,v) then (rev-path ps)@replace-by-UnPath xs (v,e,v') ps
else x#replace-by-UnPath xs (v,e,v') ps)

lemma (in valid-unMultigraph) del-unEdge-connectivity:
assumes connected ∃ ps. valid-graph.is-path (del-unEdge v e v' G) v ps v'
shows valid-unMultigraph.connected (del-unEdge v e v' G)
proof −
have valid-unMulti: valid-unMultigraph (del-unEdge v e v' G)
using valid-unMultigraph-axioms by simp
have valid-graph: valid-graph (del-unEdge v e v' G)
using valid-graph-axioms del-undirected by (metis delete-edge-valid)
obtain ex-path where ex-path: valid-graph.is-path (del-unEdge v e v' G) v ex-path v'
by (metis assms(2))
show ?thesis unfolding valid-unMultigraph.connected-def[OF valid-unMulti]
proof (rule,rule,rule)
fix n n'
assume n : n ∈ nodes (del-unEdge v e v' G)
assume n': n'∈ nodes (del-unEdge v e v' G)
assume n≠n'
obtain ps where ps:is-path n ps n'
by (metis (n≠n') n n' (connected: connected-def del-UnEdge-node))
hence valid-graph.is-path (del-unEdge v e v' G)
(n (replace-by-UnPath ps (v,e,v') ex-path) n')
proof (induct ps arbitrary:n)
case Nil
thus ?case by (metis is-path.simps(1) n' replace-by-UnPath.simps(1))
valid-graph
valid-graph.is-path-simps(1)
next
case (Cons x xs)
obtain x1 x2 x3 where x:x=(x1,x2,x3) by (metis prod-cases3)
have x=(v,e,v') ⟹ ?case
proof −
assume x=(v,e,v')
hence valid-graph.is-path (del-unEdge v e v' G)
\[(\text{replace-by-UnPath} \ (x \neq x') \ (v, e, v') \ \text{ex-path}) \ n'\]
\[= \ \text{valid-graph}.\text{is-path} \ (\text{del-edge} v \ e \ v' \ G)\]
\[n \ (\text{ex-path} \times (\text{replace-by-UnPath} \ (x \neq x') \ (v, e, v') \ \text{ex-path})) \ n'\]
\[\text{by (metis replace-by-UnPath.\text{simps}(2))}\]
\[\text{also have ...=} \ \text{True}\]
\[\text{by (metis Cons.hyps Cons.prems } (x = (v, e, v')) \ \text{ex-path} \ \text{is-path}.\text{simps}(2)\]
\[\text{valid-graph}\]
\[\text{valid-graph}.\text{is-path-split}\]
\[\text{finally show } ?\text{thesis by simp}\]
\[\text{qed}\]
\[\text{moreover have } x = (v', e, v) \Rightarrow ?\text{case}\]
\[\text{proof –}\]
\[\text{assume } x = (v', e, v)\]
\[\text{hence } \text{valid-graph}.\text{is-path} \ (\text{del-edge} v \ e \ v' \ G)\]
\[n \ (\text{replace-by-UnPath} \ (x \neq x') \ (v, e, v') \ \text{ex-path}) \ n'\]
\[= \ \text{valid-graph}.\text{is-path} \ (\text{del-edge} v \ e \ v' \ G)\]
\[n \ ((\text{rev-path} \ \text{ex-path}) \times (\text{replace-by-UnPath} \ (x \neq x') \ (v, e, v') \ \text{ex-path})) \ n'\]
\[\text{by (metis Cons.prems is-path.\text{simps}(2) no-id replace-by-UnPath.\text{simps}(2))}\]
\[\text{also have ...=} \ \text{True}\]
\[\text{by (metis Cons.hyps Cons.prems } (x = (v', e, v)) \ \text{is-path}.\text{simps}(2)\]
\[\text{ex-path valid-graph}\]
\[\text{valid-graph}.\text{is-path-split valid-unMulti valid-unMultigraph.\text{is-path-rev}}\]
\[\text{finally show } ?\text{thesis by simp}\]
\[\text{qed}\]
\[\text{moreover have } x \neq (v', e, v) \land x \neq (v', e, v) \Rightarrow ?\text{case}\]
\[\text{by (metis Cons.hyps Cons.prems del-\text{UnEdge-frame} is-path.\text{simps}(2) replace-by-UnPath.\text{simps}(2) x)\]
\[\text{ultimately show } ?\text{case by auto}\]
\[\text{qed}\]
\[\text{thus } \exists \text{ps. valid-graph}.\text{is-path} \ (\text{del-edge} v \ e \ v' \ G) \ n \ \text{ps} \ n' \ \text{by auto}\]
\[\text{qed}\]
\[\text{lemma (in valid-unMultigraph) path-between-odds:}\]
\[\text{assumes } \text{odd}(\text{degree} v \ G) \ \text{odd}(\text{degree} v' \ G) \ \text{finite} \ E \ v \neq v' \ \text{num-of-odd-nodes} G = 2\]
\[\text{shows } \exists \text{ps. iso-trail} v \ \text{ps} \ v'\]
\[\text{proof –}\]
\[\text{have } v \in V\]
\[\text{proof (rule ccontr)}\]
\[\text{assume } v \notin V\]
\[\text{hence } \forall e \in E, \text{fst e } \neq v \ \text{by (metis E-valid(1) imageI set-mp)}\]
\[\text{hence } \text{degree} v \ G = 0 \ \text{unfolding degree-def using (finite E)}\]
\[\text{by force}\]
\[\text{thus False using (odd}(\text{degree} v \ G)) \ \text{by auto}\]
\[\text{qed}\]
\[\text{have } v' \in V\]
\[\text{proof (rule ccontr)}\]
\[\text{assume } v' \notin V\]
hence $\forall e \in E, \text{fst } e \neq v'$ by (metis $E$-valid(1) imageI set-mp)

hence $\text{degree } v' G = 0$ unfolding degree-def using (finite $E$

by force

thus False using (odd(\text{degree } v' G)); by auto

qed

then obtain max-path $v_0$ where max-path:

is-trail $v_0$ max-path $v'$

($\forall n. \forall w \in E. \neg$is-trail $n$ (w#max-path) $v'$)

using longest-path[of $v'$] by (metis assms(3))

have even(\text{degree } v_0 G)\implies v_0 = v' \implies v_0 = v

by (metis assms(2))

moreover have even(\text{degree } v_0 G)\implies v_0 = v' \implies v_0 = v

proof
− assume even(\text{degree } v_0 G) v_0 \neq v'

hence $\exists v, v_1. \text{is-trail}$

$v_1 (w#\text{max-path}) v'$

by (metis assms(3) extend-distinct-path max-path(1))

thus thesis by (metis (full-types) is-trail.simps(2) max-path(2) prod.exhaust)

qed

moreover have odd(\text{degree } v_0 G)\implies v_0 = v' \implies v_0 = v

proof
− assume odd(\text{degree } v_0 G) v_0 = v

hence $\exists v, v_1. \text{is-trail}$

$v_1 (w#\text{max-path}) v'$

by (metis assms(3) extend-distinct-path max-path(1))

thus thesis by (metis (full-types) List.set.simps(2) insert-subset max-path(2)

path-in-edges)

qed

moreover have odd(\text{degree } v_0 G)\implies v_0 \neq v' \implies v_0 = v

proof (rule ccontr)
− assume $v_0 \neq v$ odd(\text{degree } v_0 G) v_0 \neq v'

moreover have $v \in$ odd-nodes-set $G$

using ($v \in V$ : odd (\text{degree } v G)) unfolding odd-nodes-set-def

by auto

moreover have $v' \in$ odd-nodes-set $G$

using ($v' \in V$ : odd (\text{degree } v' G)) unfolding odd-nodes-set-def

by auto

ultimately have $\{v, v', v_0\} \subseteq$ odd-nodes-set $G$

using is-path-memb[of is-trail-intro[OF is-trail v_0 max-path v']] max-path(1)

unfolding odd-nodes-set-def

by auto

moreover have card $\{v, v', v_0\} = 3$ using $(v_0 \neq v) \land (v \neq v') \land (v_0 \neq v')$ by auto

moreover have finite (odd-nodes-set $G$


by auto

ultimately have $3 \leq$ card(odd-nodes-set $G$) by (metis card mono)

thus False using (\text{num-of-odd-nodes } G = 2); unfolding num-of-odd-nodes-def

by auto
proof
ultimately have \( v \equiv v \) by auto
thus \(?thesis\) by (metis max-path(1))
qed

lemma (in valid-unMultigraph) del-unEdge-even-connectivity:
assumes finite E finite V connected \( \forall n \in V. \) even\((\text{degree } n \in G) (v, e, v') \in E\)
shows valid-unMultigraph\connected (\( \text{del-unEdge } v \in v' \in G \))
proof –
  have valid-unMulti\validMultigraph (\( \text{del-unEdge } v \in v' \in G \))
    using valid-unMultigraph-axioms by simp
  have valid-graph: valid-graph (\( \text{del-unEdge } v \in v' \in G \))
    using valid-graph-axioms del-undirected by (metis delete-edge-valid)
  have \( \text{fin-E: } \text{finite}(\text{edges } (\text{del-unEdge } v \in v' \in G)) \)
    by (metis (hide-tams, no-types) assms(1) del-undirected delete-edge-def
      finite-Diff select-convs(2))
  have \( \text{fin-V': } \text{finite}(\text{nodes } (\text{del-unEdge } v \in v' \in G)) \)
    by (metis (mono-tags) assms(2) del-undirected delete-edge-def select-convs(1))
  have all-even: \( \forall n \in \text{nodes } (\text{del-unEdge } v \in v' \in G). \) \( n \neq \{v, v'\} \)
    \( \rightarrow \) \( \text{even}(\text{degree } n \in (\text{del-unEdge } v \in v' \in G)) \)
    by (metis (full-types) assms(4) del-unEdge-even-connectivity(1) assms(4) assms(5))
  moreover have \( \text{even } (\text{degree } v \in G) \) by (metis (full-types) \text{E-validD}(1) assms(4) \text{assms}(5))
  moreover have num-of-odd-nodes \( G = 0 \)
    using \( \forall n \in V. \) even\((\text{degree } n \in G) \)\( \rightarrow \) finite V
    unfolding num-of-odd-nodes-def odd-nodes-set-def by auto
  ultimately have num-of-odd-nodes \( (\text{del-unEdge } v \in v' \in G) = 2 \)
    using del-UnEdge-even-even[of G v e v',OF valid-unMultigraph-axioms]
    by (metis assms(1) assms(2) \text{assms}(5) \
     \text{monoid-add-class}, add.left-neutral)
  moreover have \( \text{odd } \) (\( \text{degree } v \) \( \in \) \( \text{del-unEdge } v \in v' \in G))
    using \( \text{even } (\text{degree } v \in G) \)\( \rightarrow \) \( \text{del-UnEdge-even}\}\( \langle v, e, v' \rangle \in E \)\( \rightarrow \) \( \text{finite } E \)
    unfolding odd-nodes-set-def
    by auto
  moreover have \( \text{odd } \) (\( \text{degree } v' \) \( \in \) \( \text{del-unEdge } v \in v' \in G))
    using \( \text{even } (\text{degree } v' \in G) \)\( \rightarrow \) \( \text{del-UnEdge-even}\}\( \langle v, e, v' \rangle \in E \)\( \rightarrow \) \( \text{finite } E \)
    unfolding odd-nodes-set-def
    by auto
  moreover have \( \text{finite } \) (\( \text{edges } (\text{del-unEdge } v \in v' \in G) \))
    using \( \text{finite } E \)\( \rightarrow \)
    by auto
  moreover have \( v \neq v' \) using no-id \( \langle v, e, v' \rangle \in E \)\( \rightarrow \)
  ultimately have \( \exists ps. \) valid-unMultigraph\is\trail \( \langle \text{del-unEdge } v \in v' \in G \rangle v ps v' \)
    using valid-unMultigraph\is\trail \( \langle \text{del-unMulti, } of \rangle v v' \)
    by auto
  thus \(?thesis\) by (metis (full-types) \text{assms}(3) del-unEdge-connectivity valid-unMulti
    valid-unMultigraph\is\trail-intro)
qed
lemma (in valid-graph) path-end:ps\not=[\cdot] \implies is-path v ps v' \implies v' = snd (snd(last ps))
  by (induct ps arbitrary:v,auto)

lemma (in valid-unMultigraph) connectivity-split:
  assumes connected \sim\sim valid-unMultigraph.connected (del-unEdge v w v' G)
  \( (v,w,v') \in E \)
  obtains G1 G2 where
    nodes G1=\{n. \exists ps. valid-graph.is-path (del-unEdge v w v' G) n ps v\}
    and edges G1=\{(n,e,n'). (n,e,n')\in edges (del-unEdge v w v' G)
                   \land n\in nodes G1 \land n'\in nodes G1\}
    nodes G2=\{n. \exists ps. valid-graph.is-path (del-unEdge v w v' G) n ps v'\}
    and edges G2=\{(n,e,n'). (n,e,n')\in edges (del-unEdge v w v' G)
                   \land n\in nodes G2 \land n'\in nodes G2\}
    and nodes G1 \cup edges G2 = edges (del-unEdge v w v' G)
    and edges G1 \cap edges G2=\{\}
    and nodes G1 \cup nodes G2=nodes (del-unEdge v w v' G)
    and nodes G1 \cap nodes G2=\{\}
    and valid-unMultigraph G1
    and valid-unMultigraph G2
    and valid-unMultigraph.connected G1
    and valid-unMultigraph.connected G2

proof –
  have valid0:valid-graph (del-unEdge v w v' G) using valid-graph-axioms
    by (metis del-undirected delete-edge-valid)
  have valid0':valid-unMultigraph (del-unEdge v w v' G) using valid-unMultigraph-axioms
    by (metis del-unEdge-valid)
  obtain G1-nodes where G1-nodes:G1-nodes=
    \{n. \exists ps. valid-graph.is-path (del-unEdge v w v' G) n ps v\}
    by metis
  then obtain G1 where G1:G1=
    \{(n\in G1-nodes, edges={\{(n,e,n'). (n,e,n')\in edges (del-unEdge v w v' G)
                              \land n\in G1-nodes \land n'\in G1-nodes\}}\}
    by metis
  obtain G2-nodes where G2-nodes:G2-nodes=
    \{n. \exists ps. valid-graph.is-path (del-unEdge v w v' G) n ps v'\}
    by metis
  then obtain G2 where G2:G2=
    \{(n\in G2-nodes, edges={\{(n,e,n'). (n,e,n')\in edges (del-unEdge v w v' G)
                              \land n\in G2-nodes \land n'\in G2-nodes\}}\}
    by metis
  have valid-G1:valid-unMultigraph G1
    using G1 valid-unMultigraph.corres[OF valid0'] valid-unMultigraph.no-id[OF valid0']
    by (unfold-locales,auto)
  hence valid-G1:valid-graph G1 using valid-unMultigraph-def by auto
  have valid-G2:valid-unMultigraph G2
using $G_2$ valid-unMultigraph.corres[OF valid0] valid-unMultigraph.no-id[OF valid0]
by (unfold-locales, auto)
hence valid-$G_2'$: valid-graph $G_2$ using valid-unMultigraph-def by auto
have nodes $G_1\{n. \exists ps. \text{valid-graph.is-path (del-unEdge } v \; w \; v') n \; ps \; v\}$
  using $G_1$-nodes $G_1$ by auto
moreover have edges $G_1\{(n,e,n'). (n,e,n')\in\text{edges (del-unEdge } v \; w \; v') \land n\in\text{nodes } G_1 \land n'\in\text{nodes } G_1\}$
  using $G_1$-nodes $G_1$ by auto
moreover have nodes $G_2\{n. \exists ps. \text{valid-graph.is-path (del-unEdge } v \; w \; v') n \; ps \; v'\}$
  using $G_2$-nodes $G_2$ by auto
moreover have nodes $G_1 \cup \text{nodes } G_2 = \text{nodes (del-unEdge } v \; w \; v')$
proof (rule ccontr)
  assume nodes $G_1 \cup \text{nodes } G_2 \neq \text{nodes (del-unEdge } v \; w \; v')$
  moreover have nodes $G_1 \subseteq \text{nodes (del-unEdge } v \; w \; v')$
    using valid-graph.is-path-memb[OF valid0] $G_1$ $G_1$-nodes by auto
  moreover have nodes $G_2 \subseteq \text{nodes (del-unEdge } v \; w \; v')$
    using valid-graph.is-path-memb[OF valid0] $G_2$ $G_2$-nodes by auto
ultimately obtain $n$ where $n$:
  n\in\text{nodes (del-unEdge } v \; w \; v') \land n\notin\text{nodes } G_1 \land n\notin\text{nodes } G_2$
  by auto
hence $n$-neg-v : $\neg(\exists ps. \text{valid-graph.is-path (del-unEdge } v \; w \; v') n \; ps \; v')$
and
  $n$-neg-v': $\neg(\exists ps. \text{valid-graph.is-path (del-unEdge } v \; w \; v') n \; ps \; v')$
using $G_1$ $G_1$-nodes $G_2$ $G_2$-nodes by auto
hence $n\notin v$ by (metis $n$($I$) valid0 valid-graph.is-path-simps($I$))
then obtain ps \ where ps: is-path $n$ ps $v$ using \ connected;
  by (metis $E$-validD($I$) assms($3$) connected-def del-UnEdge-node $n$($I$))
then obtain $n'$ \ where ps': ps' = takeWhile ($\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)$)
  by auto
moreover have $n$-neg-p': $n$-neg-p = $n$-neg-p \ & dropWhile ($\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)$)
  by auto
ultimately obtain $n'$ \ where is-path-p': is-path $n$ ps' $n'$
  and is-path $n'$ (dropWhile ($\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)$) ps) $v$
  using ps is-path-split[of $n$ ps' dropWhile ($\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)$)]
  by auto
have $n' = v \lor n' = v'$
proof (cases dropWhile ($\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)$) ps)
  case Nil
  hence ps' = ps' using ps-ps' by (metis append-Nil2)
  hence $n' = v$ using ps is-path-ps' path-end by (metis (mono-tags)
    is-path.split($I$))
thus \?thesis by auto
next
\[
\text{case (Cons } x \text{ xs)}
\]
\[\text{hence dropWhile } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)) \text{ nvs } \not\in [] \text{ by auto}
\]
\[\text{hence } \text{hd (dropWhile } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)) \text{ nvs}) = (v, w, v')
\]
\[\text{\quad \lor \text{hd (dropWhile } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)) \text{ nvs}) = (v', w, v')
\]
\[\text{by (metis (lifting, full-types) hd-dropWhile)}
\]
\[\text{hence } x = (v, w, v') \lor x = (v', w, v) \text{ using Cons by auto}
\]
\[\text{thus } \text{thesis}
\]
\[\text{using } \text{is-path } n' \text{ (dropWhile } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)) \text{ nvs)}
\]
\[\text{by (metis Cons is-path.simps(2))}
\]
\[\text{qed}
\]
\[\text{moreover have valid-graph.is-path } (\text{del-unEdge } v \text{ w } v' \text{ G}) \text{ n nvs' n'}
\]
\[\text{using is-path-nvs' nvs'}
\]
\[\text{proof (induct nvs' arbitrary:n nvs)}
\]
\[\text{case Nil}
\]
\[\text{thus } ?\text{case } \text{by (metis del-UnEdge-node is-path.simps(1) valid0 valid-graph.is-path.simps(1))}
\]
\[\text{next}
\]
\[\text{case (Cons } x \text{ xs)}
\]
\[\text{obtain } x1 x2 x3 \text{ where } x : = (x1, x2, x3) \text{ by (metis prod-cases3)}
\]
\[\text{hence is-path } x3 \text{ x n'} \text{ using Cons by auto}
\]
\[\text{moreover have } x \text{ s } = \text{takeWhile } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)) \text{ (tl
\]
\[\text{usng } (x \neq x \text{ s } = \text{takeWhile } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)) \text{ nvs)}
\]
\[\text{by (metis (lifting, no-types) append-Cons list.distinct(1) takeWhile.simps(2))}
\]
\[\text{takeWhile-dropWhile-id list.sel(3))}
\]
\[\text{ultimately have valid-graph.is-path } (\text{del-unEdge } v \text{ w } v' \text{ G}) \text{ x3 x n'}
\]
\[\text{using Cons by auto}
\]
\[\text{moreover have } x \neq (v, w, v') \land x \neq (v', w, v)
\]
\[\text{using Cons(3) set-takeWhileD[of x } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v))
\]
\[\text{ris uis)]}
\]
\[\text{by (metis List.set.simps(2) insertII)}
\]
\[\text{hence } x \in \text{edges } (\text{del-unEdge } v \text{ w } v' \text{ G})
\]
\[\text{by (metis Cons.prems(1) del-UnEdge-frame is-path.simps(2) x)}
\]
\[\text{ultimately show } ?\text{case using x}
\]
\[\text{by (metis Cons.prems(1) is-path.simps(2) valid0 valid-graph.is-path.simps(2))}
\]
\[\text{qed}
\]
\[\text{ultimately show False using n-neg-v n-neg-v' by auto}
\]
\[\text{qed}
\]
\[\text{moreover have nodes G1 } \cap \text{ nodes G2 } = \{}
\]
\[\text{proof (rule ccontr)}
\]
\[\text{assume nodes G1 } \cap \text{ nodes G2 } \not= \{}
\]
\[\text{then obtain } n \text{ where } n : \text{n\in nodes G1 n\in nodes G2 by auto}
\]
\[\text{then obtain nvs n'vs where}
\]
\[\text{nvs : valid-graph.is-path } (\text{del-unEdge } v \text{ w } v' \text{ G}) \text{ n nvs v and}
\]
\[\text{n'vs : valid-graph.is-path } (\text{del-unEdge } v \text{ w } v' \text{ G}) \text{ n n'vs v'}
\]
\[\text{using G1 G2 G1-nodes G2-nodes by auto}
\]
\[\text{hence valid-graph.is-path } (\text{del-unEdge } v \text{ w } v' \text{ G}) \text{ v ((rev-path nvs)@n'vs') v'}
\]
\[\text{using valid-unMultigraph.is-path-rev}[\text{OF valid0'}] \text{ valid-graph.is-path-split}[\text{OF
\]
\[39
\]
valid0]

by auto

hence valid-unMultigraph.connected (del-unEdge v w v' G)

by (metis assms(1) del-unEdge-connectivity)

thus False by (metis assms(2))

qed

moreover have edges G1 ∪ edges G2 = edges (del-unEdge v w v' G)

proof (rule ccontr)

assume edges G1 ∪ edges G2 ≠ edges (del-unEdge v w v' G)

moreover have edges G1 ⊆ edges (del-unEdge v w v' G) using G1 by auto

moreover have edges G2 ⊆ edges (del-unEdge v w v' G) using G2 by auto

ultimately obtain n e n' where

nen':
\[(n,e,n') ∈ edges (del-unEdge v w v' G)\]
\[(n,e,n') ∉ edges G1 \quad (n,e,n') ∉ edges G2\]

by auto

moreover have n∈nodes (del-unEdge v w v' G)

by (metis nen'(1) valid0 valid-graph.E-validD(1))

moreover have n'∈nodes (del-unEdge v w v' G)

by (metis nen'(1) valid0 valid-graph.E-validD(2))

ultimately have \((n∈nodes G1 ∧ n'∈nodes G2) ∨ (n∈nodes G2 ∧ n'∈nodes G1)\)

using G1 G2 (nodes G1 ∪ nodes G2 = nodes (del-unEdge v w v' G)) by auto

moreover have n∈nodes G1 → n'∈nodes G2 → False

proof —

assume n∈nodes G1 n'∈nodes G2

then obtain n vs n' vs where

n vs : valid-graph.is-path (del-unEdge v w v' G) n n vs v and
n' vs : valid-graph.is-path (del-unEdge v w v' G) n' n' vs v'

using G1 G2 G1-nodes G2-nodes by auto

hence valid-graph.is-path (del-unEdge v w v' G) v

\(((rev-path n vs)@(n,e,n')#n' vs)\) v'

using valid-multigraph.is-path-rev[OF valid0'] valid-graph.is-path-split[OF valid0]

\{(n,e,n') ∈ edges (del-unEdge v w v' G)\}

by auto

hence valid-unMultigraph.connected (del-unEdge v w v' G)

by (metis assms(1) del-unEdge-connectivity)

thus False by (metis assms(2))

qed

moreover have n∈nodes G2 → n'∈nodes G1 → False

proof —

assume n'∈nodes G1 n∈nodes G2

then obtain n' vs n vs where

n' vs : valid-graph.is-path (del-unEdge v w v' G) n' n' vs v and
n vs : valid-graph.is-path (del-unEdge v w v' G) n n vs v'

using G1 G2 G1-nodes G2-nodes by auto

moreover have \((n',e,n) ∈ edges (del-unEdge v w v' G)\)

by (metis nen'(1) valid0' valid-unMultigraph.corres)
ultimately have valid-graph.is-path (del-unEdge v w v') v
  ((rev-path n'’es)@(n’,e,n)#nes) v'
using valid-unMultigraph.is-path-rev[OF valid0] valid-graph.is-path-split'[OF
valid0]
  by auto
hence valid-unMultigraph.connected (del-unEdge v w v')
  by (metis assms(1) del-unEdge-connectivity)
thus False by (metis assms(2))
qed
ultimately show False by auto
qed
moreover have edges G1 ∩ edges G2 = {}
proof (rule ccontr)
  assume edges G1 ∩ edges G2 ≠ {}
then obtain n e n' where (n,e,n')∈edges G1 (n,e,n')∈edges G2 by auto
hence n∈nodes G1 n∈nodes G2 using G1 G2 by auto
thus False using ⟨nodes G1 ∩ nodes G2={}⟩ by auto
qed
moreover have valid-unMultigraph.connected G1
unfolding valid-unMultigraph.connected-def[OF valid-G1]
proof (rule,rule,rule)
  fix n n'
  assume n : n ∈ nodes G1
  assume n': n'∈ nodes G1
  assume n≠n'
  obtain ps where valid-graph.is-path (del-unEdge v w v') n ps v
    using G1 G1-nodes Cons
    by (metis prod-cases3)
  moreover have (x1,x2,x3)∈edges (del-unEdge v w v')
    by (metis Cons.prems valid0 valid-graph.is-path.simps(2) x)
  ultimately show ?case
    by (metis valid0 valid-G1 valid-unMultigraph.is-trail.simps(1)
      valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)
next
  case (Cons x xs)
  obtain x1 x2 x3 where x:x =(x1,x2,x3) by (metis prod-cases3)
  have x1∈nodes G1 using G1 G1-nodes Cons.prems x
  by (metis (lifting) mem-Collect-eq select-convs(1)
    valid-graph.is-path.simps(1))
  ultimately show ?case
    by (metis valid0 valid-G1 valid-unMultigraph.is-trail.simps(1)
      valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)

by (metis Cons.prems valid0 valid-graph.is-path.simps(2) x)
hence valid-graph.is-path G1 z3 xs v using Cons.hyps by auto
moreover have x1=n by (metis Cons.prems valid0 valid-graph.is-path.simps(2) x)
ultimately show ?case using x valid-G1' by (metis valid-graph.is-path.simps(2))

qed
obtain ps' where valid-graph.is-path (del-unEdge v w v') n' ps' v
using G1 G1-nodes n' by auto
hence ps':valid-graph.is-path G1 n' ps' v
proof (induct ps' arbitrary:n')
case Nil
moreover have v@nodes G1 using G1 G1-nodes valid0
by (metis (lifting, no-types) calculation mem-Collect-eq select-convs(1)
valid-graph.is-path.simps(1))
ultimately show ?case
by (metis valid0 valid-G1 valid-unMultigraph.is-trail.simps(1)
valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)
next
case (Cons x xs)
obtain x1 x2 x3 where x:x=(x1,x2,x3) by (metis prod-cases3)
have x1∈nodes G1 using G1 G1-nodes Cons.prems x
by (metis (lifting) mem-Collect-eq select-convs(1) valid0 valid-graph.is-path.simps(2))
moreover have (x1,x2,x3)∈edges (del-unEdge v w v') G)
by (metis Cons.prems valid0 valid-graph.is-path.simps(2) x)
ultimately show ?case
by (metis valid0 valid-G1 valid-unMultigraph.is-trail.simps(1)
valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)

moreover have x1=x2.x3 where x:x=(x1,x2,x3) by (metis prod-cases3)
have x1∈nodes G1 using G1 G1-nodes Cons.prems x
by (metis (lifting) mem-Collect-eq select-convs(1) valid0 valid-graph.is-path.simps(2))
moreover have (x1,x2,x3)∈edges (del-unEdge v w v' G)
by (metis Cons.prems valid0 valid-graph.is-path.simps(2) x)
ultimately show ?case
by (metis valid0 valid-G1 valid-unMultigraph.is-trail.simps(1)
valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)

by (metis Cons.prems valid0 valid-graph.is-path.simps(2))
next
case Nil
ultimately show ?case using x valid-G1' by (metis valid-graph.is-path.simps(2))

qed

moreover have valid-unMultigraph.connected G2
unfolding valid-unMultigraph.connected-def[OF valid-G2]
proof (rule,rule,rule)
fix n n'

42
assume \( n : n \in \text{nodes} \ G2 \)
assume \( n': n' \in \text{nodes} \ G2 \)
assume \( n \neq n' \)
obtain \( ps \) \text{ where } valid-graph.is-path \((\text{del-unEdge} \ v \ w \ v') \ G \ n \ ps \ v' \)
using \( G2 \) \text{G2-nodes} \ n \ \text{by} \ \text{auto} 

hence \( ps : \text{valid-graph.is-path} \ G2 \ n \ ps \ v' \)

proof \( (\text{induct} \ ps \ \text{arbitrary:n}) \)
    case Nil
    moreover have \( v' \in \text{nodes} \ G2 \) \text{ using } \( G2 \) \text{G2-nodes} \ valid0
    by \( (\text{metis} \ \text{lifting, no-types} \ \text{calculation} \ \text{mem-Collect-eq} \ \text{select-convs}(1) \ valid-graph.is-path.simps(1)) \)

ultimately show \( \text{?case} \)
        by \( (\text{metis} \ valid0 \ valid-G2 \ valid-multigraph.is-trail.simps(1) \ valid-graph.is-path.simps(1) \ valid-multigraph.is-trail-intro) \)

next
    case \( (\text{Cons} \ x \ xs) \)
obtain \( x1 \ x2 \ x3 \) \text{ where } \( x : x = (x1, x2, x3) \) \text{ by } \( (\text{metis} \ \text{prod-cases3}) \)
have \( x1 \in \text{nodes} \ G2 \) \text{ using } \( G2 \) \text{G2-nodes} \ Cons.prems \ x
by \( (\text{metis} \ \text{lifting} \ \text{mem-Collect-eq} \ \text{select-convs}(1) \ valid0 \ valid-graph.is-path.simps(2)) \)

moreover have \( (x1, x2, x3) \in \text{edges} \ (\text{del-unEdge} \ v \ w \ v') \ G \)
by \( (\text{metis} \ Cons.prems \ valid0 \ valid-graph.is-path.simps(2) \ x) \)

ultimately have \( (x1, x2, x3) \in \text{edges} \ G2 \)
using \( \text{nodes} \ G1 \cap \text{nodes} \ G2 = \{\} \ \text{edges} \ G1 \cup \text{edges} \ G2 = \text{edges} \ (\text{del-unEdge} \ v \ w \ v' \ G) \)

by \( (\text{metis} \ \text{IntI} \ \text{Un-iff} \ \text{assms}(1) \ \text{bex-empty} \ \text{connected-def} \ \text{del-UnEdge-node} \ valid0 \ valid0') \)
valid-G1' valid-graph.E-validD(1) valid-graph.E-validD(2) valid-multigraph.no-id

moreover have \( \text{valid-graph.is-path} \ (\text{del-unEdge} \ v \ w \ v' \ G) \ x \ x \ x \ v' \)
by \( (\text{metis} \ Cons.prems \ valid0 \ valid-graph.is-path.simps(2) \ x) \)

hence \( \text{valid-graph.is-path} \ G2 \ x3 \ x \ v' \) \text{ using } Cons.hyps \ \text{by} \ \text{auto} 

moreover have \( x1 = n \) \text{ by } \( (\text{metis} \ Cons.prems \ valid0 \ valid-graph.is-path.simps(2) \ x) \)

ultimately show \( \text{?case} \) \text{ using } \( x \) \text{ valid-G2'} \text{ by } \( (\text{metis} \ valid-graph.is-path.simps(2)) \)

qed

obtain \( p s' \) \text{ where } valid-graph.is-path \( (\text{del-unEdge} \ v \ w \ v' \ G) \ n' \ ps' \ v' \)
using \( G2 \) \text{G2-nodes} \ n' \ \text{by} \ \text{auto} 

hence \( ps' : \text{valid-graph.is-path} \ G2 \ n' \ ps' \ v' \)

proof \( (\text{induct} \ ps' \ \text{arbitrary:n'}) \)
    case Nil
    moreover have \( v' \in \text{nodes} \ G2 \) \text{ using } \( G2 \) \text{G2-nodes} \ valid0
    by \( (\text{metis} \ \text{lifting, no-types} \ \text{calculation} \ \text{mem-Collect-eq} \ \text{select-convs}(1) \ valid-graph.is-path.simps(1)) \)

ultimately show \( \text{?case} \)
by \( (\text{metis} \ valid0 \ valid-G2 \ valid-multigraph.is-trail.simps(1) \ valid-graph.is-path.simps(1) \ valid-multigraph.is-trail-intro) \)

next
    case \( (\text{Cons} \ x \ xs) \)
obtain \( x1 \ x2 \ x3 \) \text{ where } \( x : x = (x1, x2, x3) \) \text{ by } \( (\text{metis} \ \text{prod-cases3}) \)
have \( x_1 \in \text{nodes } G_2 \) using \( G_2 \)-nodes Cons.prems \( x \)
by (metis (lifting) mem-Collect-eq select-cons (1) valid0 valid-graph.is-path.simps (2))
moreover have \( (x_1, x_2, x_3) \in \text{edges } (\text{del-unEdge } v \ w \ v') G_2 \)
by (metis Cons.prems valid0 valid-graph.is-path.simps (2) \( x \))
ultimately have \( (x_1, x_2, x_3) \in \text{edges } G_2 \)
using \( \langle \text{nodes } G_1 \cap \text{nodes } G_2 = \{\} \rangle \langle \text{edges } G_1 \cup \text{edges } G_2 = \text{edges} (\text{del-unEdge } v \ w \ v') G_2 \rangle \)
by (metis IntI Un-iff assms (1) bex-empty connected-def del-UnEdge-node valid0 valid0' valid-G1' valid-graph.E-validD (1) valid-graph.E-validD (2) valid-unMultigraph.no-id)
moreover have valid-graph.is-path (\( \text{del-unEdge } v \ w \ v' G_2 \) \( x \) \( x_3 \) \( v' \))
by (metis Cons.prems valid0 valid-graph.is-path.simps (2) \( x \))
hence valid-graph.is-path \( G_2 \) \( x \) \( x_3 \) \( v' \) using Cons.hyps by auto
moreover have \( x_1 = n' \) by (metis Cons.prems valid0 valid-graph.is-path.simps (2) \( x \))
ultimately show \( \exists \text{case using } x \text{ valid-G2' by (metis valid-graph.is-path.simps (2))} \)
qed

hence valid-graph.is-path \( G_2 \) \( v' \) (rev-path \( ps' \)) \( n' \)
using valid-unMultigraph.is-path-rev[OF valid-G2]
by auto
hence valid-graph.is-path \( G_2 \) \( n \) (\( ps@\)(rev-path \( ps' \))) \( n' \)
using \( ps \) valid-graph.is-path-split[OF valid-G2',of \( n \) \( ps \) rev-path \( ps' \) \( n' \)]
by auto
thus \( \exists ps. \text{ valid-graph.is-path } G_2 \) \( n \) \( ps \) \( n' \) by auto
qed
ultimately show \( \exists \text{thesis using } \text{valid-G1 valid-G2 that by auto} \)
qed

**Lemma sub-graph-degree-frame:**
assumes valid-graph \( G_2 \) edges \( G_1 \cup \text{edges } G_2 = \text{edges } G \) \( \text{nodes } G_1 \cap \text{nodes } G_2 = \{\} \) \( n \in \text{nodes } G_1 \)
shows degree \( n \) \( G = \text{degree } n \) \( G_1 \)
proof –
have \( \{ e \in \text{edges } G. \text{ fst } e = n \} \subseteq \{ e \in \text{edges } G_1. \text{ fst } e = n \} \)
proof
fix \( e \) assume \( e \in \{ e \in \text{edges } G. \text{ fst } e = n \} \)
hence \( e \in \text{edges } G \) \( \text{ fst } e = n \) by auto
moreover have \( n \notin \text{nodes } G_2 \)
using \( \langle \text{nodes } G_1 \cap \text{nodes } G_2 = \{\} \rangle \langle n \in \text{nodes } G_1 \rangle \)
by auto
hence \( e \notin \text{edges } G_2 \) using valid-graph.E-validD[OF valid-graph G2][\( \text{fst } e = n \)]
by (metis PairE fst-conv)
ultimately have \( e \in \text{edges } G_1 \) using \( \text{edges } G_1 \cup \text{edges } G_2 = \text{edges } G \) by auto
thus \( e \in \{ e \in \text{edges } G_1. \text{ fst } e = n \} \) using \( \text{fst } e = n \) by auto
qed
moreover have \{ e \in \text{edges } G_1. \text{fst } e = n \} \subseteq \{ e \in \text{edges } G. \text{fst } e = n \} 
by \{(\text{metis } \text{lifting}) \text{ Collect-mono } \text{Un-iff } \text{assms(2)}\)
ultimately show \(?\text{thesis unfolding degree-def by auto}\)
qed

lemma odd-nodes-no-edge[simp]: finite \((\text{nodes } g) \implies \text{num-of-odd-nodes } (g \{\text{edges:}=}\{\}) = 0\)
unfolding num-of-odd-nodes-def odd-nodes-set-def degree-def by simp

4 Adjacent nodes

definition (in valid-unMultigraph) adjacent:: 'v \Rightarrow 'v \Rightarrow bool where
adjacent v v' \equiv \exists w. (v,w,v')\in E

lemma (in valid-unMultigraph) adjacent-sym: adjacent v v' \iff adjacent v' v
unfolding adjacent-def by auto

lemma (in valid-unMultigraph) adjacent-no-loop[simp]: adjacent v v' \implies v \neq v'
unfolding adjacent-def by auto

lemma (in valid-unMultigraph) adjacent-V[simp]:
assumes adjacent v v'
shows v\in V v'\in V
using assms E-validD unfolding adjacent-def by auto

lemma (in valid-unMultigraph) adjacent-finite:
finite E \implies finite \{ n. adjacent v n \}
proof -
assume finite E
\{ fix S v 
  have finite S \implies finite \{ n. \exists w. (v,w,n)\in S \}
  proof \((\text{induct } S \text{ rule: finite-induct})\)
    case empty
    thus ?case by auto
    next
    case \(\text{insert } x F\)
    obtain x1 x2 x3 where x:=(x1,x2,x3) by \{(\text{metis } \text{prod-cases3})\}
    have x1=v \implies ?case
    proof -
      assume x1=v
      hence \{ n. \exists w. (v, w, n) \in \text{insert } x F \} = \text{insert } x3 \{ n. \exists w. (v, w, n) \in F \}
      using x by auto
      thus ?thesis using insert by auto
    qed
    moreover have x1\neq v \implies ?case
    proof -
      assume x1\neq v

45
hence \( \{ n. \exists w. (v, w, n) \in \text{insert } x F \} = \{ n. \exists w. (v, w, n) \in F \} \) using \( x \) by auto

thus \(?thesis\) using \( \text{insert}\) by auto

qed

ultimately show \(?case\) by auto

qed

note \( aux=\text{this} \)

show \(?thesis\) using \( aux\)[OF \( \text{finite } E\), of \( v\)] unfolding \( \text{adjacent-def}\) by auto

qed

5 Undirected simple graph

locale \( \text{valid-unSimpGraph}=\text{valid-unMultigraph } G \) for \( G::(\'v, \'w ) \text{ graph}+ \)

assumes \( \text{no-multi[simp]}:\ (v, w, u) \in \text{edges } G \implies (v, w, u) \in \text{edges } G \)

\( \implies w = w' \)

lemma (in \( \text{valid-unSimpGraph}\)) \( \text{finV-to-finE}\)[simp]:

assumes \( \text{finite } V \)

shows \( \text{finite } E \)

proof (cases \( \{ (v1, v2). \text{adjacent } v1 \ v2\} = \{\} \))

case True

hence \( E=\{\} \) unfolding \( \text{adjacent-def}\) by auto

thus \( \text{finite } E\) by auto

next

case False

have \( \{ (v1, v2). \text{adjacent } v1 \ v2\} \subseteq V \times V \) using \( \text{adjacent-V}\) by auto

moreover have \( \text{finite } (V \times V) \) using \( \langle \text{finite } V\rangle \) by auto

ultimately have \( \text{finite } \{ (v1, v2). \text{adjacent } v1 \ v2\} \) using \( \text{finite-subset}\) by auto

hence \( \text{card } \{ (v1, v2). \text{adjacent } v1 \ v2\} \neq 0 \) using \( \text{False}\) card-eq-0-iff by auto

moreover have \( \text{card } E=\text{card } \{ (v1, v2). \text{adjacent } v1 \ v2\} \)

proof

have \( \lambda(v1,w,v2). (v1,v2))'E = \{ (v1,v2). \text{adjacent } v1 \ v2\} \)

proof

have \( \forall x. x \in (\lambda(v1,w,v2). (v1,v2))'E \implies x \in \{ (v1,v2). \text{adjacent } v1 \ v2\} \)

unfolding \( \text{adjacent-def}\) by auto

moreover have \( \forall x. x \in \{ (v1,v2). \text{adjacent } v1 \ v2\} \implies x \in (\lambda(v1,w,v2). (v1,v2))'E \)

unfolding \( \text{adjacent-def}\) by force

ultimately show \(?thesis\) by force

qed

moreover have \( \text{inj-on } (\lambda(v1,w,v2). (v1,v2)) E \) unfolding \( \text{inj-on-def}\) by auto

ultimately show \(?thesis\) by (metis \( \text{card-image}\))

qed

ultimately show \( \text{finite } E\) by (metis \( \text{card-infinite}\))

qed

lemma \( \text{del-unEdge-valid}[\text{simp}]::\text{valid-unSimpGraph } G \implies \)
valid-unSimpGraph (del-unEdge \(v\ w\ u\ G\))

proof
  --
  assume valid-unSimpGraph \(G\)
  hence valid-unMultigraph (del-unEdge \(v\ w\ u\ G\))
    using valid-unSimpGraph-def [of \(G\)] del-unEdge-valid [of \(G\)] by auto
  moreover have valid-unSimpGraph-axioms (del-unEdge \(v\ w\ u\ G\))
    using valid-unSimpGraph-no-multi [OF valid-unSimpGraph-def \(G\)]
    unfolding valid-unSimpGraph-axioms-def del-unEdge-def by auto
  ultimately show valid-unSimpGraph (del-unEdge \(v\ w\ u\ G\))
    using valid-unSimpGraph-def by auto
  qed

lemma (in valid-unSimpGraph) del-UnEdge-non-adj:
  \((v, w, u) \in E \implies \neg valid-unMultigraph.adjacent (del-unEdge \(v\ w\ u\ G\))\)

proof
  assume \((v, w, u) \in E\)
  and ccontr: valid-unMultigraph.adjacent (del-unEdge \(v\ w\ u\ G\)) \(v\ u\)
  have valid: valid-unMultigraph (del-unEdge \(v\ w\ u\ G\))
    using valid-unMultigraph-axioms by auto
  then obtain \(w'\) where \((v, w, w', u) \in \text{edges} \(G\))
    using ccontr unfolding valid-unMultigraph.adjacent-def [OF valid] by auto
  hence \(w' \neq w\) by auto
  moreover have \((v, w', u) \in E\)
    using vw'u unfolding del-unEdge-def by auto
  ultimately show False using no-multi [of \(v\ w\ u\ w'\)] \((v, w, u) \in E\) by auto
  qed

lemma (in valid-unSimpGraph) degree-adjacent: finite \(E\) \(\implies\) degree \(v\ G\) = card \(\{n. adjacent v n\}\)

proof (induct degree \(v\ G\) arbitrary: \(G\))
  case 0
  note valid3 = valid-unSimpGraph \(G\);
  hence valid2: valid-unMultigraph \(G\) using valid-unSimpGraph-def by auto
  have \(\{a. valid-unMultigraph.adjacent \(G\ v\ a\}\} = \{\}\)
    proof (rule ccontr)
      assume \(\{a. valid-unMultigraph.adjacent \(\!G\ v\ a\}\} \neq \{\}\)
      then obtain \(w\ u\) where \((v, w, u) \in \text{edges} \(G\))
        unfolding valid-unMultigraph.adjacent-def [OF valid2] by auto
      hence degree \(v\ G\) \(\neq 0\) using \(\text{finite} \,(\text{edges} \(G\))\) unfolding degree-def by auto
      thus False using \(0 = \text{degree} \(v\ G\)\) by auto
    qed
  thus ?case by (metis 0.hyps card-empty)
next
  case (Suc \(n\))
  hence \(\{e \in \text{edges} \(G\). \text{fst} e = v\} \neq \{\}\) using card-empty unfolding degree-def by force
  then obtain \(w\ u\) where \((v, w, u) \in \text{edges} \(G\)) by auto
  have valid: valid-unMultigraph \(G\) using valid-unSimpGraph-def \(G\) valid-unSimpGraph-def

47
by auto

hence valid'; valid-unMultigraph (del-unEdge v w u G) by auto
have valid-unSimpGraph (del-unEdge v w u G)
  using del-unEdge-valid' (valid-unSimpGraph G) by auto
moreover have \( n = \text{degree} \ v \ (\text{del-unEdge} \ v \ w \ u \ G) \)
  using \( \text{Suc} \ n = \text{degree} \ v \ G \vdash (v, w, u) \in \text{edges} \ G \) \ del-edge-undirected-degree-plus[of \( G \ v \ w \ u \)]
  by (metis Suc.prems(1) Suc-eq-plus1 diff-Suc-1 valid valid-unMultigraph)
moreover have finite (edges (del-unEdge v w u G))
  using (finite (edges G)) unfolding del-unEdge-def by auto
ultimately have \( \text{degree} \ v \ (\text{del-unEdge} \ v \ w \ u \ G) = \text{card} \ \{ n. \ \text{valid-unMultigraph}.\text{adjacent} (\text{del-unEdge} \ v \ w \ u \ G) \ v \ n \} \)
  using valid-unMultigraph.adjacent-def[OF valid]
proof –
  have \( \{ n. \ \text{valid-unMultigraph}.\text{adjacent} (\text{del-unEdge} \ v \ w \ u \ G) \ v \ n \} \subseteq \{ n. \ \text{valid-unMultigraph}.\text{adjacent} G \ v \ n \} \)
    using del-unEdge-def[of v w u G]
    unfolding valid-unMultigraph.adjacent-def[OF valid]
    valid-unMultigraph.adjacent-def[OF valid]
    by auto
moreover have \( u \in \{ n. \ \text{valid-unMultigraph}.\text{adjacent} G \ v \ n \} \)
  using \( (v,w,u)\in\text{edges} \ G \) unfolding valid-unMultigraph.adjacent-def[of valid]
  by auto
ultimately have \( \{ n. \ \text{valid-unMultigraph}.\text{adjacent} (\text{del-unEdge} \ v \ w \ u \ G) \ v \ n \} \cup \{ u \} \subseteq \{ n. \ \text{valid-unMultigraph}.\text{adjacent} G \ v \ n \} \)
  by auto
moreover have \( \{ n. \ \text{valid-unMultigraph}.\text{adjacent} G \ v \ n \} - \{ u \} \subseteq\)
  \( \{ n. \ \text{valid-unMultigraph}.\text{adjacent} (\text{del-unEdge} \ v \ w \ u \ G) \ v \ n \} \)
  using del-unEdge-def[of v w u G]
  unfolding valid-unMultigraph.adjacent-def[of valid]
  valid-unMultigraph.adjacent-def[OF valid]
  by auto
ultimately have \( \{ n. \ \text{valid-unMultigraph}.\text{adjacent} (\text{del-unEdge} \ v \ w \ u \ G) \ v \ n \} \cup \{ u \} = \{ n. \ \text{valid-unMultigraph}.\text{adjacent} G \ v \ n \} \)
  by auto
moreover have \( \text{finite} \ \{ n. \ \text{valid-unMultigraph}.\text{adjacent} G \ v \ n \} \)
  using valid-unMultigraph.adjacent-finite[OF valid (finite (edges G))]
  by simp
ultimately show \( \text{thesis} \)
by (metis Un-insert-right card-insert-disjoint finite-Un sup-bot-right)
qed
ultimately show ?case by (metis Suc.hyps(2) \( n = \text{degree} v (\text{del-unEdge} v w u G) \))
qed
end

theory KoenigsbergBridge imports MoreGraph Map Enum begin

6 Definition of Eulerian trails and circuits

definition (in valid-unMultigraph) is-Eulerian-trail:: '\( v \Rightarrow (v',w) \) path\( \Rightarrow v' \Rightarrow \) bool where
is-Eulerian-trail v v'\( \equiv \) is-trail v ps v' \( \land \) edges (rem-unPath ps G) = {}

definition (in valid-unMultigraph) is-Eulerian-circuit:: '\( v \Rightarrow (v',w) \) path\( \Rightarrow v \Rightarrow \) bool where
is-Eulerian-circuit v v'\( \equiv \) (v=v') \( \land \) (is-Eulerian-trail v ps v')

7 Necessary conditions for Eulerian trails and circuits

lemma (in valid-unMultigraph) euclerian-rev:
is-Eulerian-trail v' (rev-path ps) v\( \equiv \)is-Eulerian-trail v ps v'
proof
  have is-trail v' (rev-path ps) v\( \equiv \)is-trail v ps v'
    by (metis is-trail-rev)
  moreover have edges (rem-unPath (rev-path ps G)=edges (rem-unPath ps G)
    by (metis rem-unPath-graph)
  ultimately show ?thesis unfolding is-Eulerian-trail-def by auto
qed

theorem (in valid-unMultigraph) euclerian-cycle-ex:
assumes is-Eulerian-circuit v ps v' finite V finite E
shows \( \forall v \in V. \) even (degree v G)
proof
  obtain v ps v' where cycle:is-Eulerian-circuit v ps v' using assms by auto
  hence edges (rem-unPath ps G) = {}
    unfolding is-Eulerian-circuit-def is-Eulerian-trail-def
    by simp
  moreover have nodes (rem-unPath ps G)=nodes G by auto
  ultimately have rem-unPath ps G = G \( \langle \text{edges:=\{\}} \rangle \) by auto
  hence num-of-odd-nodes (rem-unPath ps G) = 0 by (metis assms(2) odd-nodes-no-edge)
  moreover have v=v'

49
by (metis (is-Eulerian-circuit v ps v' is-Eulerian-circuit-def))

hence num-of-odd-nodes (rem-unPath ps G) = num-of-odd-nodes G
by (metis assms(2) assms(3) cycle is-Eulerian-circuit-def)

ultimately have num-of-odd-nodes G = 0 by auto

moreover have finite (odd-nodes-set G)
using ⟨finite V; unfolding odd-nodes-set-def by auto⟩

ultimately have odd-nodes-set G = {} unfolding num-of-odd-nodes-def by auto

thus ?thesis unfolding odd-nodes-set-def by auto
qed

theorem (in valid-unMultigraph) euclerian-path-ex:
assumes is-Eulerian-trail v ps v' finite V finite E
shows (∀ v ∈ V. even (degree v G)) ∨ (num-of-odd-nodes G = 2)
proof –

obtain v ps v' where path: is-Eulerian-trail v ps v' using assms by auto

hence edges (rem-unPath ps G) = {} unfolding is-Eulerian-trail-def by simp

moreover have nodes (rem-unPath ps G) = nodes G by auto

ultimately have rem-unPath ps G = G ⟨edges:={}⟩ by auto

by (metis odd-nodes-no-edge)

have v ≠ v' ⇒ ?thesis

proof (cases even (degree v G))

  case True
  assume v ≠ v'
  have is-trail v ps v' by (metis is-Eulerian-trail-def path)
  hence num-of-odd-nodes (rem-unPath ps G) = num-of-odd-nodes G
      + (if even (degree v G) then 2 else 0)
      using rem-UnPath-even True ⟨finite V; finite E; v ≠ v'⟩ by auto
  hence num-of-odd-nodes G + (if even (degree v G) then 2 else 0) = 0
      using odd-nodes-by auto
  hence num-of-odd-nodes G = 0 by auto
  moreover have finite (odd-nodes-set G)
      using ⟨finite V; unfolding odd-nodes-set-def by auto⟩
  ultimately have odd-nodes-set G = {} unfolding num-of-odd-nodes-def by auto
  thus ?thesis unfolding odd-nodes-set-def by auto

next

  case False
  assume v ≠ v'
  have is-trail v ps v' by (metis is-Eulerian-trail-def path)
  hence num-of-odd-nodes (rem-unPath ps G) = num-of-odd-nodes G
      + (if odd (degree v G) then −2 else 0)
      using rem-UnPath-odd False ⟨finite V; finite E; v ≠ v'⟩ by auto
  hence odd-nodes-if: num-of-odd-nodes G + (if odd (degree v G) then −2 else 0)
using odd-nodes by auto
have odd (degree v G) \implies \text{thesis}
  proof -
  assume odd (degree v G)
  hence num-of-odd-nodes G = 2 using odd-nodes-if by auto
  thus \text{thesis} by simp
  qed
moreover have even (degree v G) \implies \text{thesis}
  proof -
  assume even (degree v G)
  hence num-of-odd-nodes G = 0 using odd-nodes-if by auto
  moreover have finite (odd-nodes-set G)
    using (finite V) unfolding odd-nodes-set-def by auto
  ultimately have odd-nodes-set G = \{\} unfolding num-of-odd-nodes-def
by auto
  thus \text{thesis} unfolding odd-nodes-set-def by auto
  qed
ultimately show \text{thesis} by auto
  qed
moreover have v = v' \implies \text{thesis}
  by (metis assms (2) assms (3) euclerian-cycle-ex is-Eulerian-circuit-def path)
ultimately show \text{thesis} by auto
  qed

8 Specific case of the Konigsberg Bridge Problem

datatype kon-node = a | b | c | d

datatype kon-bridge = ab1 | ab2 | ac1 | ac2 | ad1 | bd1 | cd1

definition kon-graph :: (kon-node, kon-bridge) graph where
  kon-graph ≡ \{(nodes = \{a, b, c, d\},
  edges = \{(a, ab1, b), (b, ab1, a),
    (a, ab2, b), (b, ab2, a),
    (a, ac1, c), (c, ac1, a),
    (a, ac2, c), (c, ac2, a),
    (a, ad1, d), (d, ad1, a),
    (b, bd1, d), (d, bd1, b),
    (c, cd1, d), (d, cd1, c)\}\}

instantiation kon-node :: enum
begin
definition [simp]: enum-class.enum = [a, b, c, d]
definition [simp]: enum-class.enum-all P \iff P \land P \land P \land P \land P 
definition [simp]: enum-class.enum-ex P \iff P \lor P \lor P \lor P \lor P
instance proof qed (auto, (case_tac x, auto)+)
end
instantiation \texttt{kon-bridge :: enum}

\begin{verbatim}
begin

definition \texttt{[simp]:enum-class.enum} = [ab1, ab2, ac1, ac2, ad1, cd1, bd1]
definition \texttt{[simp]:enum-class.enum-all P} =\[P\ al1 \land P\ ac1 \land P\ ac2
\land P\ ad1 \land P\ bd1\]
definition \texttt{[simp]:enum-class.enum-ex P} =\[P\ ab1 \lor P\ ab2 \lor P\ ac1 \lor P\ ac2
\lor P\ ad1 \lor P\ bd1 \lor P\ cd1\]
instance proof qed (auto,(case_tac \textit{x},auto)+)
end
\end{verbatim}

interpretation \texttt{kon-graph: valid-unMultigraph kon-graph}

\begin{verbatim}
proof (unfold-locales)
show \texttt{fst ' edges kon-graph \subseteq nodes kon-graph} by eval
next
show \texttt{snd ' snd ' edges kon-graph \subseteq nodes kon-graph} by eval
next
have \forall v\ w\ u\'. ((v, w, u') \in \texttt{edges kon-graph}) = ((u', w, v) \in \texttt{edges kon-graph})
  by eval
  thus \forall v\ w\ u\'. ((v, w, u') \in \texttt{edges kon-graph}) = ((u', w, v) \in \texttt{edges kon-graph})
    by simp
next
have \forall v\ w. (v, w, v) \notin \texttt{edges kon-graph} by eval
  thus \forall v\ w. (v, w, v) \notin \texttt{edges kon-graph} by simp
qed
\end{verbatim}

\begin{verbatim}
theorem \texttt{\neg kon-graph.is-Eulerian-trail v1 \ p v2}
proof
assume \texttt{kon-graph.is-Eulerian-trail v1 \ p v2}
moreover have \texttt{finite (nodes kon-graph)} by (metis \texttt{finite-code})
moreover have \texttt{finite (edges kon-graph)} by (metis \texttt{finite-code})
ultimately have contra:
  \((\forall v\ \in \texttt{nodes kon-graph}.\ \texttt{even (degree v kon-graph)}) \lor (\texttt{num-of-odd-nodes kon-graph} = 2)\)
    by (metis \texttt{kon-graph.euclerian-path-ex})
  have \texttt{odd(degree a kon-graph)} by eval
moreover have \texttt{odd(degree b kon-graph)} by eval
moreover have \texttt{odd(degree c kon-graph)} by eval
moreover have \texttt{odd(degree d kon-graph)} by eval
ultimately have \texttt{\neg(n-odd-nodes kon-graph = 2)} by eval
moreover have \texttt{\neg(\forall v\ \in \texttt{nodes kon-graph}.\ \texttt{even (degree v kon-graph)})} by eval
ultimately show False using contra by auto
qed
\end{verbatim}

52
9 Sufficient conditions for Eulerian trails and circuits

lemma (in valid-unMultigraph) eulerian-cons:

assumes
valid-unMultigraph.is-Eulerian-trail (del-unEdge v0 w v1 G) v1 ps v2
(v0,w,v1)∈ E

shows is-Eulerian-trail v0 ((v0,w,v1)#ps) v2

proof –

have valid:valid-unMultigraph (del-unEdge v0 w v1 G)
  using valid-unMultigraph-axioms by auto
hence distinct:valid-unMultigraph.is-trail (del-unEdge v0 w v1 G) v1 ps v2
  using assms unfolding valid-unMultigraph.is-Eulerian-trail-def [OF valid]
  by auto
hence set ps ⊆ edges (del-unEdge v0 w v1 G)
  using valid-unMultigraph.path-in-edges [OF valid] by auto
moreover have (v0,w,v1)∈ edges (del-unEdge v0 w v1 G)
  unfolding del-unEdge-def by auto
moreover have (v1,w,v0)∈ edges (del-unEdge v0 w v1 G)
  unfolding del-unEdge-def by auto
ultimately have (v0,w,v1)∈ edges (v1,w,v0)∈ edges by auto
moreover have is-trail v1 ps v2
  using distinct-path-intro [OF distinct] .
ultimately have is-trail v0 ((v0,w,v1)#ps) v2
  using (v0,w,v1)∈ E by auto
moreover have edges (rem-unPath ps (del-unEdge v0 w v1 G)) ={}
  using assms unfolding valid-unMultigraph.is-Eulerian-trail-def [OF valid]
  by auto
hence edges (rem-unPath ((v0,w,v1)#ps) G) ={}
  by (metis rem-unPath.simps(2))
ultimately show ?thesis unfolding is-Eulerian-trail-def by auto
qed

lemma (in valid-unMultigraph) eulerian-cons':

assumes
valid-unMultigraph.is-Eulerian-trail (del-unEdge v2 w v3 G) v1 ps v2
(v2,w,v3)∈ E

shows is-Eulerian-trail v1 (ps@[v2,w,v3]) v3

proof –

have valid:valid-unMultigraph (del-unEdge v3 w v2 G)
  using valid-unMultigraph-axioms del-unEdge-valid by auto
have del-unEdge v2 w v3 G=del-unEdge v3 w v2 G
  by (metis delete-edge-sym)
hence valid-unMultigraph.is-Eulerian-trail (del-unEdge v3 w v2 G) v2
  (rev-path ps) v1 using assms valid-unMultigraph.eulerian-rev [OF valid]
  by auto
hence is-Eulerian-trail v3 ((v3,w,v2)#(rev-path ps)) v1
  using eulerian-cons by (metis assms(2) corres)
hence is-Eulerian-trail v1 (rev-path((v3,w,v2)#(rev-path ps))) v3

53
using euclerian-rev by auto
moreover have rev-path([(v3,w,v2)#(rev-path ps)]) = rev-path(rev-path ps)#[(v2,w,v3)]

unfolding rev-path-def by auto
hence rev-path([(v3,w,v2)#(rev-path ps)]) = ps#[[(v2,w,v3)] by auto
ultimately show thesis by auto qed

lemma eulerian-split:
assumes nodes G1 ∩ nodes G2 = {}
edges G1 ∩ edges G2 = {}
valid-unMultigraph G1 valid-unMultigraph G2
valid-unMultigraph.is-Eulerian-trail G1 v1 ps1 v1'
valid-unMultigraph.is-Eulerian-trail G2 v2 ps2 v2'
shows valid-unMultigraph.is-Eulerian-trail (nodes=nodes G1 ∪ nodes G2,
edges=edges G1 ∪ edges G2 ∪ {v1',w,v2),(v2,w,v1')}) v1 (ps1@((v1',w,v2)#ps2)
proof –
have valid-graph G1 using valid-unMultigraph G1; valid-unMultigraph-def by auto
have valid-graph G2 using valid-unMultigraph G2; valid-unMultigraph-def by auto
obtain G where G:G=(!nodes=nodes G1 ∪ nodes G2,
edges=edges G1 ∪ edges G2
∪ {(v1',w,v2),(v2,w,v1')})
by metis
have v1'∈nodes G1
by (metis (full-types) valid-graph G1; assms(3) assms(5) valid-graph.is-path-memb
valid-unMultigraph.is-trail-intro valid-unMultigraph.is-Eulerian-trail-def)
moreover have v2∈nodes G2
by (metis (full-types) valid-graph G2; assms(4) assms(6) valid-graph.is-path-memb
valid-unMultigraph.is-trail-intro valid-unMultigraph.is-Eulerian-trail-def)
ultimately have valid-unMultigraph (nodes=nodes G1 ∪ nodes G2,
edges=edges G1 ∪ edges G2
∪ {(v1',w,v2),(v2,w,v1')})
using
valid-unMultigraph.corres[OF valid-unMultigraph G1]
valid-unMultigraph.no-id[OF valid-unMultigraph G1]
valid-unMultigraph.corres[OF valid-unMultigraph G2]
valid-unMultigraph.no-id[OF valid-unMultigraph G2]
valid-graph.E-validD[OF valid-graph G1]
valid-graph.E-validD[OF valid-graph G2]
(nodes G1 ∩ nodes G2 = {})
proof (unfold-locales,auto)
fix aa ab ba
assume (aa, ab, ba) ∈ edges G1
thus ba ∈ nodes G1 by (metis \( v' \in v. (v, e, v') \in edges G1 \implies v' \in nodes G1 \))
next

fix aa ab ba
assume ba \notin nodes G2 \ (aa, ab, ba) \in edges G2
thus ba \in nodes G1 by (metis valid-graph G2 valid-graph.E-validD(2))

qed

hence valid: valid-unMultigraph G using G by auto

hence valid':valid-graph G using valid-unMultigraph-def by auto

moreover have valid-unMultigraph.is-trail G v1 (ps1\oplus((v1',w,v2)#ps2)) v2'

proof -

have ps1-G:valid-unMultigraph.is-trail G v1 ps1 v1'

proof -

have valid-unMultigraph.is-trail G1 v1 ps1 v1' using assms
by (metis valid-unMultigraph.is-Eulerian-trail-def)
moreover have edges G1 \subseteq edges G by (metis G UnI1 Un-assoc
select-convs(2) subrelI)
moreover have nodes G1 \subseteq nodes G by (metis G inf-sup-absorb le-iff-inf
select-convs(1))
ultimately show \?
thesis using distinct-path-subset[of G1 G,OF valid-unMultigraph G1: valid]

by auto

qed

have ps2-G:valid-unMultigraph.is-trail G v2 ps2 v2'

proof -

have valid-unMultigraph.is-trail G2 v2 ps2 v2' using assms
by (metis valid-unMultigraph.is-Eulerian-trail-def)
moreover have edges G2 \subseteq edges G by (metis G inf-sup-ord(3) le-supE
select-convs(2))
moreover have nodes G2 \subseteq nodes G by (metis G inf-sup-ord(4)
select-convs(1))
ultimately show \?

by auto

qed

have valid-graph.is-path G v1 (ps1\oplus((v1',w,v2)#ps2)) v2'

proof -

have valid-graph.is-path G v1 ps1 v1'
by (metis ps1-G valid valid-unMultigraph.is-trail-intro)
moreover have valid-graph.is-path G v2 ps2 v2'
by (metis ps2-G valid valid-unMultigraph.is-trail-intro)
moreover have (v1',w,v2) \in edges G
using G by auto
ultimately show \?
thesis using valid-graph.is-path-split[of G valid',of v1 ps1 v1' w v2 ps2 v2'] by
auto

qed

moreover have distinct (ps1\oplus((v1',w,v2)#ps2))

proof -

have distinct ps1 by (metis ps1-G valid valid-unMultigraph.is-trail-path)
moreover have distinct ps2
by (metis ps2-G valid valid-unMultigraph.is-trail-path)
moreover have set ps1 ∩ set ps2 = {}
proof –
  have set ps1 ⊆ edges G1
    by (metis assms(3) assms(5) valid-unMultigraph.is-Eulerian-trail-def valid-unMultigraph.path-in-edges)
moreover have set ps2 ⊆ edges G2
    by (metis assms(4) assms(6) valid-unMultigraph.is-Eulerian-trail-def valid-unMultigraph.path-in-edges)
ultimately show ?thesis using ⟨edges G1 ∩ edges G2={}⟩ by auto
qed
moreover have (v1′,w,v2) /∈ edges G1
  using ⟨v2 ∈ nodes G2; valid-graph G1⟩
  by (metis Int-iff all-not-in-conv assms(1) valid-graph.E-validD(2))
hence (v1′,w,v2) /∈ set ps1
  by (metis (full-types) assms(3) assms(5) subsetD valid-unMultigraph path-in-edges valid-unMultigraph.is-Eulerian-trail-def )
moreover have (v1′,w,v2) /∈ edges G2
  using ⟨v1′ ∈ nodes G1; valid-graph G2⟩
  by (metis assms(1) disjoint-iff-not-equal valid-graph.E-validD(1))
hence (v1′,w,v2) /∈ set ps2
  by (metis (full-types) assms(4) assms(6) in-mono valid-unMultigraph.path-in-edges valid-unMultigraph.is-Eulerian-trail-def )
ultimately show ?thesis using distinct-append by auto
qed
moreover have set (ps1@(v1′,w,v2)#ps2)) ∩ set (rev-path (ps1@(v1′,w,v2)#ps2))) = {}
proof –
  have set ps1 ∩ set (rev-path ps1) = {}
  by (metis ps1-G valid valid-unMultigraph.is-trail-path)
moreover have set (rev-path ps2) ⊆ edges G2
  by (metis assms(4) assms(6) valid-unMultigraph.is-trail-rev valid-unMultigraph.is-Eulerian-trail-def valid-unMultigraph.path-in-edges)
hence set ps1 ∩ set (rev-path ps2) = {}
  using assms
  valid-unMultigraph.path-in-edges[OF (valid-unMultigraph G1), of v1 ps1]
  valid-unMultigraph.path-in-edges[OF (valid-unMultigraph G2), of v2 ps2]
unfolding valid-unMultigraph.is-Eulerian-trail-def[OF (valid-unMultigraph G1)]
valid-unMultigraph.is-Eulerian-trail-def[OF (valid-unMultigraph G2)]
  by auto
moreover have set ps2 ∩ set (rev-path ps2) = {}
  by (metis ps2-G valid valid-unMultigraph.is-trail-path)
moreover have set (rev-path ps1) ⊆ edges G1
  by (metis assms(3) assms(5) valid-unMultigraph.is-Eulerian-trail-def valid-unMultigraph.path-in-edges valid-unMultigraph.euclerian-rev)
hence \( \text{set } ps2 \cap \text{set } (\text{rev-path } ps1) = \{\} \)

by (metis calculation(2) distinct-append distinct-rev-path ps1-G ps2-G rev-path-append

\( \text{rev-path-double valid valid-unMultigraph.is-trail-path} \))

moreover have \((v_2,w,v_1') \notin \text{set } (ps1 \oplus ((v_1',w,v_2) \# ps2))\)

proof -

have \((v_2,w,v_1') \notin \text{edges } G1\)

using \(v_2 \in \text{nodes } G2\) \(\langle \text{valid-graph } G1\rangle\)

by (metis Int-iff all-not-in-conv assms(1) valid-graph.E-validD(1))

hence \((v_2,w,v_1') \notin \text{set } ps1\)

by (metis assms(3) assms(5) split-list valid-unMultigraph.is-trail-split')

valid-unMultigraph.is-Eulerian-trail-def)

moreover have \((v_2,w,v_1') \notin \text{edges } G2\)

using \(v_1' \in \text{nodes } G1\) \(\langle \text{valid-graph } G2\rangle\)

by (metis IntI assms(1) empty-iff valid-graph.E-validD(2))

hence \((v_2,w,v_1') \notin \text{set } ps2\)

by (metis (full-types) assms(4) assms(6) in-mono valid-unMultigraph.path-in-edges

valid-unMultigraph.is-Eulerian-trail-def)

moreover have \((v_2,w,v_1') \neq (v_1',w,v_2)\)

using \(v_1' \in \text{nodes } G1\) \(v_2 \in \text{nodes } G2\)

by (metis IntI Pair-inject assms(1) assms(5) bex-empty)

ultimately show \(?thesis\) by auto

qed

ultimately show \(?thesis\) using rev-path-append by auto

qed

ultimately show \(?thesis\) using valid-unMultigraph.is-trail-path[OF valid]

by auto

qed

moreover have \(\text{edges } (\text{rem-unPath } (ps1 \oplus ((v_1',w,v_2) \# ps2)) G) = \{\}\)

proof -

have \(\text{edges } (\text{rem-unPath } (ps1 \oplus ((v_1',w,v_2) \# ps2)) G) = \text{edges } G -

(\text{set } (ps1 \oplus ((v_1',w,v_2) \# ps2)) \cup \text{set } (\text{rev-path } (ps1 \oplus ((v_1',w,v_2) \# ps2))))\)

by (metis rem-unPath-edges)

also have ... = \text{edges } G - (\text{set } ps1 \cup \text{set } ps2 \cup \text{set } (\text{rev-path } ps1) \cup \text{set } (\text{rev-path } ps2)

\cup \{(v_1',w,v_2),(v_2,w,v_1')\}) using rev-path-append by auto

finally have \(\text{edges } (\text{rem-unPath } (ps1 \oplus ((v_1',w,v_2) \# ps2)) G) = \text{edges } G -

(\text{set } ps1 \cup

\text{set } ps2 \cup \text{set } (\text{rev-path } ps1) \cup \text{set } (\text{rev-path } ps2) \cup \{(v_1',w,v_2),(v_2,w,v_1')\}))\).

moreover have \(\text{edges } (\text{rem-unPath } ps1 G1) = \{\}\)

by (metis assms(3) assms(5) valid-unMultigraph.is-Eulerian-trail-def)

hence \(\text{edges } G1 = (\text{set } ps1 \cup \text{set } (\text{rev-path } ps1)) = \{\}\)

by (metis rem-unPath-edges)

moreover have \(\text{edges } (\text{rem-unPath } ps2 G2) = \{\}\)

by (metis assms(4) assms(6) valid-unMultigraph.is-Eulerian-trail-def)

hence \(\text{edges } G2 = (\text{set } ps2 \cup \text{set } (\text{rev-path } ps2)) = \{\}\)
by (metis rem-unPath-edges)
ultimately show ?thesis using G by auto
qed
ultimately show ?thesis by (metis G valid valid-unMultigraph.is-Eulerian-trail-def)
qed

lemma (in valid-unMultigraph) eulerian-sufficient:
assumes finite V finite E connected V \{\}
shows num-of-odd-nodes G = 2 \implies
(\exists v \in V. \exists v' \in V. \exists ps. odd(degree v G) \land odd(degree v' G) \land (v \neq v') \land is-Eulerian-trail v ps v')
and num-of-odd-nodes G = 0 \implies (\forall v \in V. \exists ps. is-Eulerian-circuit v ps v)
using (finite E) (finite V) valid-unMultigraph-axioms \{V \neq \}\ (connected)
proof (induct E arbitrary; G rule: less-induct)
case less
assume finite (edges G) and finite (nodes G) and valid-unMultigraph G and
nodes G \{\}
and valid-unMultigraph.connected G and num-of-odd-nodes G = 2
have valid-graph G using (valid-unMultigraph G) valid-unMultigraph-def by auto
obtain n1 n2 where
n1: n1 \in nodes G odd(degree n1 G)
and n2: n2 \in nodes G odd(degree n2 G)
and n1 \neq n2 unfolding num-of-odd-nodes-def odd-nodes-set-def
proof —
have \forall S. card S = 2 \implies (\exists n1 n2. n1 \in S \land n2 \in S \land n1 \neq n2)
by (metis card-eq-0-iff equals0I even-card even-numeral zero-neq-numeral)
thus ?thesis by (metis (lifting) that mem-Collect-eq)
qed
have even-except-two: \(\land \ n. \ \text{n} \in \text{nodes G} \implies n \neq n1 \implies n \neq n2 \implies \text{even}(\text{degree n G})
proof (rule ccontr)
fix n assume n \in nodes G \ n \neq n1 \ n \neq n2 odd \ (\text{degree n G})
have n \in odd-nodes-set G
by (metis (mono-tags) \ (\text{where}) \ text{nodes G} \ (\text{odd} \ (\text{degree n G})) \ 
\text{mem-Collect-eq odd-nodes-set-def})
moreover have n1 \in odd-nodes-set G
by (metis (mono-tags) mem-Collect-eq n1(1) n1(2) odd-nodes-set-def)
moreover have n2 \in odd-nodes-set G
using n2(1) n2(2) unfolding odd-nodes-set-def by auto
ultimately have \{n, n1, n2\} \subseteq odd-nodes-set G by auto
moreover have card \{n, n1, n2\} \geq 3 using \(n1 \neq n2\) \ (n \neq n1) \ (n \neq n2) by auto
moreover have finite (odd-nodes-set G)
using (finite (nodes G)) unfolding odd-nodes-set-def by auto
ultimately have card (odd-nodes-set G) \geq 3

using card-mono[of odd-nodes-set G \{n, n1, n2\}] by auto
thus False using (num-of-odd-nodes G = 2) unfolding num-of-odd-nodes-def by auto
qed
have \{ e ∈ edges G. fst e = n1\}≠{}
using n1
by (metis (full-types) degree-def empty-iff finite.emptyI odd-card)
then obtain v' w where (n1, w, v')∈edges G by auto
have v'=n2 ⇒ (3 v∈nodes G. ∃ v'∈nodes G. ∃ ps. odd (degree v G) ∧ odd (degree v' G) ∧ v ≠ v'
∧ valid-unMultigraph.is-Eulerian-trail G v ps v')
proof (cases valid-unMultigraph.connected (del-unEdge n1 w n2 G))
assume v'=n2
assume connected'.valid-unMultigraph.connected (del-unEdge n1 w n2 G)
moreover have num-of-odd-nodes (del-unEdge n1 w n2 G) = 0
using ⟨(n1, w, v') ∈ edges G ∣ finite (edges G) ∣ finite (nodes G) ∣ v' = n2⟩
⟨num-of-odd-nodes G = 2 ∣ valid-unMultigraph G ∣ del-UnEdge-odd-odd n1(2) n2(2)⟩
by force
moreover have finite (edges (del-unEdge n1 w n2 G))
using (finite (edges G)) by auto
moreover have finite (nodes (del-unEdge n1 w n2 G))
using (finite (nodes G)) by auto
moreover have edges G − \{(n1,w,n2),(n2,w,n1)\} ⊆ edges G
using Diff-iff Diff-subset ⟨(n1, w, v') ∈ edges G ∣ v' = n2⟩
by fast
hence card (edges (del-unEdge n1 w n2 G)) < card (edges G)
using (finite (edges G)) psubset-card-mono[of edges G edges G − \{(n1,w,n2),(n2,w,n1)\}]}
unfolding del-unEdge-def by auto
moreover have valid-unMultigraph (del-unEdge n1 w n2 G)
using (valid-unMultigraph G ∣ del-unEdge-valid by auto)
moreover have nodes (del-unEdge n1 w n2 G) ≠ {} by (metis (full-types) del-UnEdge-node empty-iff n1(1))
ultimately have ∀ v∈nodes (del-unEdge n1 w n2 G). ∃ ps. valid-unMultigraph.is-Eulerian-circuit
(del-unEdge n1 w n2 G) v ps v
using less.hyps[of del-unEdge n1 w n2 G] by auto
thus ?thesis using eulerian-cons
by (metis ⟨(n1, w, v') ∈ edges G ∣ n1 ≠ n2 ∣ v' = n2⟩ ∣ valid-unMultigraph G)
⟨valid-unMultigraph (del-unEdge n1 w n2 G) ∣ del-UnEdge-node n1(1) n1(2) n2(1) n2(2) ∣
valid-unMultigraph.eulerian-cons valid-unMultigraph.is-Eulerian-circuit-def⟩
next
assume v'=n2
assume not-connected:¬valid-unMultigraph.connected (del-unEdge n1 w n2 G)
have valid0:valid-unMultigraph (del-unEdge n1 w n2 G)
using \langle valid-unMultigraph G; del-unEdge-valid \rangle by auto

\textbf{hence valid0\langle valid-graph (del-unEdge n1 w n2 G) \rangle by auto}

\textbf{using valid-unMultigraph-def by auto}

\textbf{have all-even\forall n \in \text{nodes} (del-unEdge n1 w n2 G). even(\text{degree} n (del-unEdge n1 w n2 G))}

\textbf{proof –}

\textbf{have even (degree} n1 (del-unEdge n1 w n2 G))

\textbf{using} \langle n1, w, n' \rangle \in \text{edges} G \langle \text{finite} (edges G) \rangle \langle w' = n2 \rangle \langle valid-unMultigraph G \rangle n1

\textbf{by} (auto simp add: valid-unMultigraph.correct)

\textbf{moreover have even (degree} n2 (del-unEdge n1 w n2 G))

\textbf{using} \langle n1, w, n' \rangle \in \text{edges} G \langle \text{finite} (edges G) \rangle \langle w' = n2 \rangle \langle valid-unMultigraph G \rangle n2

\textbf{by} (auto simp add: valid-unMultigraph.correct)

\textbf{moreover have} \forall n. n \in \text{nodes} (del-unEdge n1 w n2 G) \implies n \neq n1 \implies n \neq n2 \implies

\textbf{even (degree} n (del-unEdge n1 w n2 G))

\textbf{using valid-unMultigraph.degree-frame[OF \langle valid-unMultigraph G, of \ n1 n2 w \rangle even-except-two}

\textbf{by} (metis no-types \langle \text{finite} (edges G) \rangle \langle del-unEdge-def empty-iff insert-iff}

\textbf{select-convs(1))}

\textbf{ultimately show} \langle \text{thesis} \rangle \text{ by auto}

\textbf{qed}

\textbf{have} \langle n1, w, n2 \rangle \in \text{edges} G \text{ by} (metis \langle n1, w, n' \rangle \in \text{edges} G \langle w' = n2 \rangle)

\textbf{hence} \langle n2, w, n1 \rangle \in \text{edges} G \text{ by} (metis valid-unMultigraph G \langle valid-unMultigraph G \rangle \langle valid-unMultigraph G \rangle)

\textbf{obtain} G1 G2 \text{ where}

G1-nodes: nodes G1=\{n. \exists ps. valid-graph.is-path (del-unEdge n1 w n2 G)

n ps n1\}

\textbf{and} G1-edges: edges G1=\{(n,e,n'). (n,e,n') \in \text{edges} (del-unEdge n1 w n2 G)

\land n \in \text{nodes} G1 \land n' \in \text{nodes} G1\}

\textbf{and} G2-nodes: nodes G2=\{n. \exists ps. valid-graph.is-path (del-unEdge n1 w n2 G)

n ps n2\}

\textbf{and} G2-edges: edges G2=\{(n,e,n'). (n,e,n') \in \text{edges} (del-unEdge n1 w n2 G)

\land n \in \text{nodes} G2 \land n' \in \text{nodes} G2\}

\textbf{and} G1-G2-edges-union: edges G1 \cup edges G2 = edges (del-unEdge n1 w n2 G)

\textbf{and} edges G1 \cap edges G2=\{\}

\textbf{and} G1-G2-nodes-union: nodes G1 \cup nodes G2=nodes (del-unEdge n1 w n2 G)

\textbf{and} nodes G1 \cap nodes G2=\{\}

\textbf{and} valid-unMultigraph G1

\textbf{and} valid-unMultigraph G2

\textbf{and} valid-unMultigraph.connected G1

\textbf{and} valid-unMultigraph.connected G2

\textbf{using} valid-unMultigraph.connectivity-split[OF \langle valid-unMultigraph G \rangle \langle valid-unMultigraph.G \rangle \langle valid-unMultigraph.connected G \rangle \langle valid-unMultigraph.connected G \rangle \langle del-unEdge G \rangle

60
have $\langle n_1, w, n_2 \rangle \in \text{edges } G \rangle$.

have edges $\langle \text{del-unEdge } n_1 w n_2 G \rangle \subset \text{edges } G$.

unfolding del-unEdge-def using $\langle n_1, w, n_2 \rangle \in \text{edges } G \rangle$ $\langle n_2, w, n_1 \rangle \in \text{edges } G \rangle$

hence $\langle \text{card } (\text{edges } G_1) < \text{card } (\text{edges } G) \rangle$ using $\langle \text{G1-G2-edges-union} \rangle$ (metis (full-types) (finite (edges G)) inf-sup-absorb less-infI2 psubset-card-mono)

moreover have finite (edges G1)

using $\langle \text{G1-G2-edges-union} \rangle$ (finite (edges G))

by (metis (full-types) (finite (edges (del-unEdge n1 w n2 G))) finite-Un less-imp-le rev-finite-subset)

moreover have nodes G1 $\subseteq$ nodes (del-unEdge n1 w n2 G)

by (metis G1-G2-nodes-union Un-upper1)

hence finite (nodes G1)

using (finite (nodes G)) (del-UnEdge-node rev-finite-subset)

by auto

moreover have n1 $\in$ nodes G1

proof −

have n1 $\in$ nodes (del-unEdge n1 w n2 G)

using (n1 $\in$ nodes G) by auto

hence valid-graph G1 using (valid-unMultigraph G2)

by (metis (full-types) (finite (edges G)) (finite (nodes G)) (finite-Union) Un-upper)

hence $\forall n \in$ nodes G1. even (degree n G1)

using (valid-unMultigraph connected G1)

by (metis (full-types) (finite (edges G)) (finite-Union) Un-upper)

thus $\langle \text{thesis using } \langle \text{G1-nodes} \rangle \rangle$

qed

hence nodes G1 $\neq \{\}$ by auto

moreover have num-of-odd-nodes G1 = 0

proof −

have valid-graph G2 using (valid-unMultigraph G2) (valid-unMultigraph-def)

by auto

hence $\forall n \in$ nodes G1. degree n G1 = degree n (del-unEdge n1 w n2 G)

using (sub-graph-degree-frame [of G2 G1] (del-unEdge n1 w n2 G))

by (metis (full-types) (finite (edges (del-unEdge n1 w n2 G))) (finite-Union) Un-upper)

hence $\forall n \in$ nodes G1. even (degree n G1)

using (valid-unMultigraph connected G1)

by (metis (full-types) (finite (edges G)) (finite-Union) Un-upper)

thus $\langle \text{thesis unfolding } \text{num-of-odd-nodes-def odd-nodes-set-def} \rangle$

by (metis (lifting) Collect-empty-eq card-eq-0-iff)

qed

ultimately have $\forall v \in$ nodes G1. $\exists ps$. valid-unMultigraph.is-Eulerian-circuit G1 v ps v

using less.hyps [of G1] (valid-unMultigraph G1) (valid-unMultigraph.connected G1)

by auto

then obtain ps1 where ps1: valid-unMultigraph.is-Eulerian-trail G1 n1 ps1 n1

using (n1 $\in$ nodes G1)

by (metis (full-types) (valid-unMultigraph G1) (valid-unMultigraph.is-Eulerian-circuit-def))

have card (edges G2) < card (edges G)

using G1-G2-edges-union (edges (del-unEdge n1 w n2 G) $\subset$ edges G)

by (metis (full-types) (finite (edges G)) inf-sup-ord le-less-trans psubset-card-mono)
moreover have \( \text{finite} \ (\text{edges} \ G_2) \)

using \( G_1-G_2\)-edges-union \( (\text{finite} \ (\text{edges} \ G)) \)
by \( \text{metis} \ (\text{edges} \ (\text{del-unEdge} \ n1 \ w \ n2 \ G) \subset \text{edges} \ G; \text{finite-Un less-imp-le rev-finite-subset}) \)

moreover have \( \text{nodes} \ G_2 \subseteq \text{nodes} \ (\text{del-unEdge} \ n1 \ w \ n2 \ G) \)
by \( \text{metis} \ G_1-G_2\)-nodes-union Un-upper2 \)
hence \( \text{finite} \ (\text{nodes} \ G_2) \)

moreover have \( n2 \in \text{nodes} \ G_2 \)
proof

have \( n2 \in \text{nodes} \ (\text{del-unEdge} \ n1 \ w \ n2 \ G) \)
using \( \text{metis} \ n2 \in \text{nodes} \ G \)

hence \( \text{valid-graph}. \text{is-path} \ (\text{del-unEdge} \ n1 \ w \ n2 \ G) \ n2 \ [\ n2 \)
using \( \text{valid0'} \) by \( \text{metis} \ \text{valid-graph}. \text{is-path-simps}(1) \)
thus \( \text{?thesis using} \ G_2\)-nodes \) by \( \text{auto} \)

moreover have \( \text{nodes} \ G_2 \neq \{\} \) by \( \text{auto} \)

moreover have \( \text{num-of-odd-nodes} \ G_2 = 0 \)

proof

have \( \text{valid-graph} \ G_1 \) using \( \text{valid-unMultigraph} \ G_1; \text{valid-unMultigraph-def} \)
by \( \text{auto} \)

hence \( \forall n \in \text{nodes} \ G_2. \text{even}(\text{degree} \ n \ G_2) \) using \( \text{all-even} \)
thus \( \text{?thesis unfolding} \ \text{num-of-odd-nodes-def odd-nodes-set-def} \)
by \( \text{metis} \ (\text{lifting}) \ \text{Collect-empty-eq card-eq-0-iff} \)

qed

ultimately have \( \forall v \in \text{nodes} \ G_2. \exists ps. \text{valid-unMultigraph.is-Eulerian-circuit} \ G_2 \ v \ ps \ v \)
using \( \text{less.hyps}[\text{of} \ G_2]: \text{valid-unMultigraph} \ G_2; \langle \text{valid-unMultigraph.connected} \ G_2 \rangle \)
by \( \text{auto} \)

then obtain \( ps2 \) where \( ps2:\text{valid-unMultigraph.is-Eulerian-trail} \ G_2 \ n2 \ ps2 \ n2 \)
using \( \langle n2 \in \text{nodes} \ G_2 \rangle \)
by \( \text{metis} \ (\text{full-types}) \ \text{valid-unMultigraph} \ G_2; \text{valid-unMultigraph.is-Eulerian-circuit-def} \)
have \( \langle \text{nodes} = \text{nodes} \ G1 \cup \text{nodes} \ G2, \text{edges} = \text{edges} \ G1 \cup \text{edges} \ G2 \cup \{ (n1, w, n2), (n2, w, n1) \}, \text{edges} = \ G \rangle \)
proof

have \( \text{edges} \ (\text{del-unEdge} \ n1 \ w \ n2 \ G) \cup \{ (n1, w, n2), (n2, w, n1) \} = \text{edges} \ G \)
using \( \langle n1,w,n2\rangle \in \text{edges} \ G; \langle n2,w,n1\rangle \in \text{edges} \ G \)
unfolding \( \text{del-unEdge-def} \) by \( \text{auto} \)

moreover have \( \text{nodes} \ (\text{del-unEdge} \ n1 \ w \ n2 \ G) = \text{nodes} \ G \)
unfolding del-unEdge-def by auto
ultimately have \(|\text{nodes} = \text{nodes} (\text{del-unEdge} n1 w n2 G), \text{edges} = \text{edges} (\text{del-unEdge} n1 w n2 G) \cup \{(n1, w, n2), (n2, w, n1)\}\) = G by auto
moreover have \(|\text{nodes} = \text{nodes} G1 \cup \text{nodes} G2, \text{edges} = \text{edges} G1 \cup \text{edges} G2 \cup \{(n1, w, n2), (n2, w, n1)\}\) = \(|\text{nodes} = \text{nodes} (\text{del-unEdge} n1 w n2 G), \text{edges} = \text{edges} (\text{del-unEdge} n1 w n2 G) \cup \{(n1, w, n2), (n2, w, n1)\}\) by (metis G1-G2-edges-union G1-G2-nodes-union)
ultimately show \(?\)thesis by auto
qed
moreover have valid-unMultigraph.is-Eulerian-trail \(|\text{nodes} = \text{nodes} G1 \cup \text{nodes} G2, \text{edges} = \text{edges} G1 \cup \text{edges} G2 \cup \{(n1, w, n2), (n2, w, n1)\}\) = n1 (ps1 @ (n1, w, n2) # ps2) n2
using eulerian-split[of G1 G2 n1 ps1 n2 ps2 n2 w]
by (metis (edges G1 \cap edges G2 = {}) \{\text{nodes} G1 \cap \text{nodes} G2 = {}\})
(valid-unMultigraph G1)
(valid-unMultigraph G2; ps1 ps2)
ultimately show \(?\)thesis by (metis \(\exists n1 \neq n2\) \(\exists n1(1) n1(2) n2(1) n2(2)\))
qed
moreover have \(v' \neq n2 \implies (\exists v \in \text{nodes} G. \exists v' \in \text{nodes} G. \exists ps. \text{odd \ (degree \ v G)} \land \text{odd \ (degree \ v' G)})\)
\(\land \ v \neq v' \land \text{valid-unMultigraph.is-Eulerian-trail} G v ps v'\)
proof (cases valid-unMultigraph.connected (del-unEdge n1 w v' G))
case True
assume \(v' \neq n2\)
assume connected':valid-unMultigraph.connected (del-unEdge n1 w v' G)
have \(n1 \in \text{nodes} (\text{del-unEdge} n1 w v' G)\) by (metis del-UnEdge-node n1)
using valid-unMultigraph.del-UnEdge-even[OF valid-unMultigraph G \(\{n1, w, v'\}\) \(\notin\) edges G]
(finite (edges G)) | odd (degree n1 G)
unfolding odd-nodes-set-def by auto
moreover have odd-n2:odd (degree n2 (del-unEdge n1 w v' G))
using valid-unMultigraph.degree-frame[OF valid-unMultigraph G \(\{\text{finite \ (edges G)}\)]]
of n2 n1 v' w \(\land \ n1 \neq n2\) \(\land \ v' \neq n2\)
by (metis empty-iff insert-iff n2(2))
moreover have even (degree v' G)
using even-except-two[of v']
by (metis (full-types) \(\{n1, w, v'\} \in \text{edges} G\) \(\langle v' \neq n2\rangle\) valid-graph G)
(valid-unMultigraph G) valid-graph.E-validD(2) valid-unMultigraph.no-id)
hence odd-v':odd (degree v' (del-unEdge n1 w v' G))
using valid-unMultigraph.del-UnEdge-even[OF valid-unMultigraph G \(\{n1, w, v'\}\) \(\notin\) edges G]
(finite (edges G))
unfolding odd-nodes-set-def by auto
ultimately have two-odds: num-of-nodes (del-unEdge n1 w v' G) = 2
by (metis (lifting) (v' ≠ n2); valid-graph G; valid-unMultigraph G;
⟨(n1, w, v') ∈ edges G; finite (edges G); finite (nodes G); num-of-odd-nodes
G = 2;
  del-UnEdge-odd-even even-except-two n1(2) valid-graph. E-validD(2))
moreover have valid0: valid-unMultigraph (del-unEdge n1 w v' G)
using del-unEdge-valid (valid-unMultigraph G) by auto
moreover have edges G = {⟨n1, w, v'⟩, (v', w, n1)⟩ ⊂ edges G
using ⟨(n1, w, v') ∈ edges G; by auto
hence card (edges (del-unEdge n1 w v' G)) < card (edges G)
using (finite (edges G)); unfolding del-unEdge-def
by (metis (hide-lams, no-types) psubset-card-mono select-convs(2))
moreover have finite (edges (del-unEdge n1 w v' G))
unfolding del-unEdge-def
by (metis (full-types); (finite (edges G)); finite-Diff select-convs(2))
moreover have finite (nodes (del-unEdge n1 w v' G))
moreover have del-unEdge-def by (metis (finite (nodes G)); select-convs(1))
moreover have nodes (del-unEdge n1 w v' G) ≠ {}
by (metis (full-types); del-UnEdge-node empty-iff n1(1))
ultimately obtain s t ps where
s: s ∈ nodes (del-unEdge n1 w v' G) odd (degree s (del-unEdge n1 w v' G))
and t: t ∈ nodes (del-unEdge n1 w v' G) odd (degree t (del-unEdge n1 w v' G))
and s ≠ t
and s-ps-t: valid-unMultigraph.is-Eulerian-trail (del-unEdge n1 w v' G) s ps t
using connected' less.hyps[of (del-unEdge n1 w v' G)] by auto
hence (s=n2 ∧ t=v' v s=v' ∧ t=n2)
using odd-n2 odd-v' two-odds (finite (edges G); valid-unMultigraph G;
by (metis (mono-tags); del-UnEdge-node empty-iff even-except-two even-n1
insert-iff
valid-unMultigraph.degree-frame)
moreover have s=n2 ⇒ t=v' ⇒ thesis
by (metis ⟨n1, w, v'⟩ ∈ edges G; n1 ≠ n2; valid-unMultigraph G; n1(1)
n1(2) n2(1) n2(2)
s-ps-t valid0 valid-unMultigraph. eulerian-rev valid-unMultigraph.eulerian-cons)
moreover have s=v' ⇒ t=n2 ⇒ thesis
by (metis ⟨n1, w, v'⟩ ∈ edges G; n1 ≠ n2; valid-unMultigraph G; n1(1)
n1(2) n2(1) n2(2)
s-ps-t valid0 valid-unMultigraph. eulerian-cons)
ultimately show thesis by auto
next
case False
assume v'≠n2
assume not-connected: valid-unMultigraph.connected (del-unEdge n1 w v' G)
have ⟨v',w,n1⟩ ∈ edges G using ⟨n1,w,v'⟩ ∈ edges G;
by (metis valid-unMultigraph G; valid-unMultigraph.corres)
have valid0: valid-unMultigraph (del-unEdge n1 w v' G)
using ⟨valid-unMultigraph G; del-unEdge-valid by auto
hence valid0; valid-graph (del-unEdge n1 w v' G)
using valid-unMultigraph-def by auto
have even-n1: even (degree n1 (del-unEdge n1 w v' G))
using valid-unMultigraph.del-UnEdge-even[OF valid-unMultigraph G]
\langle n1, w, v' \rangle \in edges G;
  \langle finite (edges G) \rangle ) \ n1
unfolding odd-nodes-set-def by auto
moreover have odd-n2: odd (degree n2 (del-unEdge n1 w v' G))
using \langle n1 \neq n2 \rangle \langle v' \neq n2 \rangle \ n2 valid-unMultigraph.degree-frame[OF valid-unMultigraph G]
\langle finite (edges G), of n2 n1 v' w \rangle
by auto
moreover have v' \neq n1
using valid-unMultigraph.no-id[OF valid-unMultigraph G] \langle n1, w, v' \rangle \in edges G;
by auto
hence odd-v': odd (degree v' (del-unEdge n1 w v' G))
using \langle v' \neq n2 \rangle even-except-two[of v']
valid-graph.E-validD2[OF valid-graph G] \langle n1, w, v' \rangle \in edges G;
valid-unMultigraph.del-UnEdge-even[OF valid-unMultigraph G] \langle n1, w, v' \rangle \in edges G;
\langle finite (edges G) \rangle
unfolding odd-nodes-set-def by auto
ultimately have even-except-two:\n. n \in nodes (del-unEdge n1 w v' G) \Longrightarrow
n \neq n2
\Longrightarrow n \neq v' \Longrightarrow even (degree n (del-unEdge n1 w v' G))
using del-UnEdge-node[of - n1 w v' G] even-except-two valid-unMultigraph.degree-frame[OF
\langle valid-unMultigraph G; \langle finite (edges G) \rangle, of - n1 v' w \rangle
by force
obtain G1 G2 where
  G1-nodes: nodes G1 = \{ n. \ \exists ps. valid-graph.is-path (del-unEdge n1 w v' G) n ps n1 \}
  and G1-edges: edges G1 = \{ (n, e, n'). (n, e, n') \in edges (del-unEdge n1 w v' G) \}
  \wedge n \in nodes G1
  \wedge n' \in nodes G1 \}
  and G2-nodes: nodes G2 = \{ n. \ \exists ps. valid-graph.is-path (del-unEdge n1 w v' G) n ps n' \}
  and G2-edges: edges G2 = \{ (n, e, n'). (n, e, n') \in edges (del-unEdge n1 w v' G) \}
  \wedge n \in nodes G2
  \wedge n' \in nodes G2 \}
  and G1-G2-edges-union: edges G1 \cup edges G2 = edges (del-unEdge n1 w v' G)
  and edges G1 \cap edges G2 = \{ \}
  and G1-G2-nodes-union: nodes G1 \cup nodes G2 = nodes (del-unEdge n1 w v' G)
  and nodes G1 \cap nodes G2 = \{ \}
  and valid-unMultigraph G1
  and valid-unMultigraph G2
  and valid-unMultigraph.connected G1
and valid-unMultigraph.connected G2
using valid-unMultigraph.connectivity-split[OF \langle valid-unMultigraph G \rangle,\langle valid-unMultigraph.connected G \rangle, not-connected \langle \{n1, w, v\}' \in \text{edges } G \rangle].

have \text{n2} \in \text{nodes } G2 \text{ using extend-distinct-path}
proof
  have finite (\text{edges } (\text{del-unEdge } n1 \ w \ v' \ G))
  unfolding del-unEdge-def using finite (\text{edges } G) by auto
  moreover have num-of-odd-nodes (\text{del-unEdge } n1 \ w \ v' \ G) = 2
  by (metis \langle \{n1, w, v\}' \in \text{edges } G \rangle, \langle \{v', w, n1\} \in \text{edges } G \rangle, \text{num-of-odd-nodes } G = 2)
  \langle v' \neq n2 \rangle \langle \text{valid-graph } G \rangle \langle \text{del-UnEdge-even-odd } \text{delete-edge-sym } \text{even-except-two \langle finite (\text{edges } G) \rangle \langle finite (\text{nodes } G) \rangle \langle \text{valid-unMultigraph } G \rangle \langle n1(2) \text{ valid-graph.E-validD(2) valid-unMultigraph.no-id } \rangle \rangle \text{ ultimately have } \exists \text{ps. } \text{valid-unMultigraph.is-trail } (\text{del-unEdge } n1 \ w \ v' \ G)
\text{n2 } ps \ v'
using valid-unMultigraph.path-between-odds[OF valid0,of \text{n2} \ v',OF odd-n2 odd-v'] (v'\neq n2]
by auto
hence \exists \text{ps. } \text{valid-graph.is-path } (\text{del-unEdge } n1 \ w \ v' \ G) \text{n2 } ps \ v'
by (metis valid0 valid-unMultigraph.is-trail-intro)
thus ?thesis using G2-nodes by auto
qed
have v' \in \text{nodes } G2
proof
  have valid-graph.is-path (\text{del-unEdge } n1 \ w \ v' \ G) v' [] v'
  by (metis (full-types) \langle \{n1, w, v\}' \in \text{edges } G \rangle \langle \text{valid-graph } G \rangle \text{del-UnEdge-node valid0} \langle \text{valid-graph.E-validD(2) valid-graph.is-path-simps(1) } \rangle
  thus ?thesis by (metis (lifting) G2-nodes mem-Collect-eq)
qed
have \text{edges-subset:edges } (\text{del-unEdge } n1 \ w \ v' \ G) \subseteq \text{edges } G
using \langle \{n1, w, v\}' \in \text{edges } G \rangle, \langle \{v', w, n1\} \in \text{edges } G \rangle
unfolding del-unEdge-def by auto
hence \text{card (edges } G1) < \text{card (edges } G)
by (metis G1-G2-edges-union inf-sup-absorb \langle \text{finite (edges } G) \rangle \ text{less-infl2 psubset-card-mono)
moreover have \text{finite (edges } G1)
by (metis (full-types) G1-G2-edges-union edges-subset finite-Un finite-subset
\langle \text{finite (edges } G) \rangle \ text{less-imp-le)
moreover have \text{finite (nodes } G1)
using G1-G2-nodes-union \langle \text{finite (nodes } G) \rangle
unfolding del-unEdge-def by (metis (full-types) finite-Un select-convs(1))
moreover have n1 \in \text{nodes } G1
proof
  have valid-graph.is-path (\text{del-unEdge } n1 \ w \ v' \ G) n1 [] n1

66
by (metis (full-types) del-UnEdge-node n1(1) valid0' valid-graph.is-path-simps(1))

thus \( \text{thesis} \) by (metis (lifting) G1-nodes mem-Collect-eq)

qed

moreover hence nodes G1 \( \neq \{\} \) by auto

moreover have num-of-odd-nodes G1 = 0

proof

  have \( \forall n \in \text{nodes G1}. \ \text{even} (\text{degree } n) \) \( \text{del-unEdge } n1 \ w \ v' G) \)
  using even-except-two' odd-v' odd-n2 \( \langle n2 \in \text{nodes G2} : \text{nodes G1} \cap \text{nodes G2} = \{\} \rangle \langle v' \in \text{nodes G2} \rangle \)
  by (metis (full-types) G1-G2-nodes-union Un-iff disjoint-iff-not-equal)

moreover have valid-graph G2

ultimately have \( \forall n \in \text{nodes G2}. \ \text{even} (\text{degree } n G2) \)

thus \( \text{thesis} \) unfolding num-of-odd-nodes-def odd-nodes-set-def

by (metis (lifting) card-eq-0-iff empty-Collect-eq)

qed

ultimately obtain ps1 where ps1: valid-unMultigraph.is-Eulerian-trail G1 n1 ps1 n1

using ⟨valid-unMultigraph G1⟩ ⟨valid-unMultigraph.connected G1⟩ less.hyps[of G1]

by (metis valid-unMultigraph.is-Eulerian-circuit-def)

have \( \text{card} (\text{edges G2}) < \text{card} (\text{edges G}) \)

by (metis G1-G2-edges-union finite (edges G) edges-subset inf-sup-absorb

less-infI2

  ps1-subset-card-mono sup-commute)

moreover have finite (edges G2)

by (metis (full-types) G1-G2-edges-union finite-Un \( \text{finite} (\text{edges G})\) less-le

rev-finite-subset)

moreover have finite \( (\text{nodes G2}) \)

by (metis (mono-tags) G1-G2-nodes-union del-UnEdge-node le-sup-iff \( \text{finite} \) (nodes G))

rev-finite-subset subsetI)

moreover have \( \text{nodes G2} \neq \{\} \) using \( \langle v' \in \text{nodes G2} \rangle \) by auto

moreover have num-of-odd-nodes G2 = 2

proof

  have \( \forall n \in \text{nodes G2}. \ n \notin \{n2,v'\}\rightarrow\text{even} (\text{degree } n) \) \( \text{del-unEdge } n1 \ w \ v' G) \)
  using even-except-two'
  by (metis (full-types) G1-G2-nodes-union Un-iff insert-iff)

moreover have valid-graph G1

using ⟨valid-unMultigraph G1⟩ valid-unMultigraph-def by auto

ultimately have \( \forall n \in \text{nodes G2}. \ n \notin \{n2,v'\}\rightarrow\text{even} (\text{degree } n G2) \)

by (metis G1-G2-edges-union Int-commute Un-commute \( \text{nodes G1} \cap \text{nodes G2} = \{\} \))

ultimately obtain ps1 where ps1: valid-unMultigraph.is-Eulerian-trail G1 n1 ps1 n1

using ⟨valid-unMultigraph G1⟩ ⟨valid-unMultigraph.connected G1⟩ less.hyps[of G1]

by (metis valid-unMultigraph.is-Eulerian-circuit-def)

have \( \text{card} (\text{edges G2}) < \text{card} (\text{edges G}) \)

by (metis G1-G2-edges-union finite (edges G) edges-subset inf-sup-absorb

less-infI2

  ps1-subset-card-mono sup-commute)

moreover have finite (edges G2)

by (metis (full-types) G1-G2-edges-union finite-Un \( \text{finite} (\text{edges G})\) less-le

rev-finite-subset)

moreover have finite \( (\text{nodes G2}) \)

by (metis (mono-tags) G1-G2-nodes-union del-UnEdge-node le-sup-iff \( \text{finite} \) (nodes G))

rev-finite-subset subsetI)

moreover have \( \text{nodes G2} \neq \{\} \) using \( \langle v' \in \text{nodes G2} \rangle \) by auto

moreover have num-of-odd-nodes G2 = 2

proof

  have \( \forall n \in \text{nodes G2}. \ n \notin \{n2,v'\}\rightarrow\text{even} (\text{degree } n) \) \( \text{del-unEdge } n1 \ w \ v' G) \)
  using even-except-two'
  by (metis (full-types) G1-G2-nodes-union Un-iff insert-iff)

moreover have valid-graph G1

using ⟨valid-unMultigraph G1⟩ valid-unMultigraph-def by auto

ultimately have \( \forall n \in \text{nodes G2}. \ n \notin \{n2,v'\}\rightarrow\text{even} (\text{degree } n G2) \)

by (metis G1-G2-edges-union Int-commute Un-commute \( \text{nodes G1} \cap \text{nodes G2} = \{\} \))
hence \( \forall n \in \text{nodes } G2. n \notin \{n2, v'\} \rightarrow n \notin \{v \in \text{nodes } G2. \text{ odd (degree } v \text{ } G2)\} \)

by (metis (lifting) mem-Collect-eq)
moreover have odd(degree n2 G2)
  using sub-graph-degree-frame[of G1 G2 del-unEdge n1 w v' G]
by (metis (hide-lams, na-types) G1-G2-edges-union \( \cap \) \text{nodes } G1 \& \text{ nodes } G2 = \{\});

\( G2 = \{\} \)
⟨valid-graph G1 \& n2 \in \text{nodes } G2\rangle inf-assoc inf-bot-right inf-sup-absorb
odd-n2 sup-bot-right sup-commute)

hence \( n2 \notin \{v \in \text{nodes } G2. \text{ odd (degree } v \text{ } G2)\} \)
by (metis (lifting) \( n2 \in \text{nodes } G2\); mem-Collect-eq)
moreover have odd(degree v' G2)
  using sub-graph-degree-frame[of G1 G2 del-unEdge n1 w v' G]
by (metis G1-G2-edges-union Int-commute Un-commute \( \cap \) \text{nodes } G1 \& \text{ nodes } G2 = \{\});

\( v' \in \text{nodes } G2. \langle \text{valid-graph } G1 \& \text{ odd-v' } \rangle \)

hence \( v' \in \{v \in \text{nodes } G2. \text{ odd (degree } v \text{ } G2)\} \)
by (metis (full-types) Collect-conj-eq Collect-mem-eq Int-Collect \( v' \in \text{nodes } G2)\))
ultimately have \( \{v \in \text{nodes } G2. \text{ odd (degree } v \text{ } G2)\} = \{n2, v'\} \)
using \( \text{finite (nodes } G2)\) by (induct G2,auto)
thus \( \exists \text{thesis using } (v' \neq n2); \)
  unfolding \( \text{num-of-odd-nodes-def odd-nodes-set-def by auto} \)
qed
ultimately obtain s t ps2 where
s: s \( \in \text{nodes } G2 \) odd (degree s G2)
and t: t \( \in \text{nodes } G2 \) odd (degree t G2)
and s \( \neq t \)
and s-ps2-t: valid-unMultigraph.is-Eulerian-trail G2 s ps2 t
using \( \langle \text{valid-unMultigraph } G2 \rangle \langle \text{valid-unMultigraph} . \text{connected } G2 \rangle \text{ less.hyps[of } G2)\)
by auto
moreover have valid-graph G1
  using \( \langle \text{valid-unMultigraph G1} \rangle \langle \text{valid-unMultigraph-def by auto} \)
ultimately have \( (s= n2 \& t= v') \lor (s= v' \& t= n2) \)
using odd-n2 odd-v' even-except-two'
sub-graph-degree-frame[of G1 G2 \{del-unEdge n1 w v' G\}]
by (metis G1-G2-edges-union G1-G2-nodes-union UnI1 \( \cap \) \text{nodes } G1 \& \text{ nodes } G2 = \{\}); inf-commute
sup-commute)

moreover have merge-G1-G2: \( \text{nodes } = \text{nodes } G1 \cup \text{nodes } G2, \text{ edges } = \text{edges } G1 \cup \text{edges } G2 \cup \)
\( \{(n1, w, v'), (v', w, n1)\}\) \( \Rightarrow G \)
proof -
have edges \( \langle \text{del-unEdge } n1 w v' G \rangle \cup \{(n1, w, v'), (v', w, n1)\} = \text{edges } G \)
using \( \langle n1, w, v' \rangle \text{edges } G \langle v', w, n1 \rangle \text{edges } G \)
unfolding del-unEdge-def by auto
moreover have nodes \( \langle \text{del-unEdge } n1 w v' G \rangle = \text{nodes } G \)
unfolding del-unEdge-def by auto
ultimately have \( \{\text{nodes} = \text{nodes} (\text{del-unEdge} \ n1 \ w \ v' \ G), \text{edges} = \text{edges} (\text{del-unEdge} \ n1 \ w \ v' \ G) \cup \{(n1, w, v'), (v', w, n1)\}\} = G \)

by auto

moreover have \( \{\text{nodes} = \text{nodes} G1 \cup \text{nodes} G2, \text{edges} = \text{edges} G1 \cup \text{edges} G2 \cup \{(n1, w, v'), (v', w, n1)\}\} = \{\text{nodes} = \text{nodes} (\text{del-unEdge} \ n1 \ w \ v' \ G), \text{edges} = \text{edges} (\text{del-unEdge} \ n1 \ w \ v' \ G) \cup \{(n1, w, v'), (v', w, n1)\}\} \)

by (metis G1-G2-edges-union G1-G2-nodes-union)

ultimately show \(?\text{thesis} by auto\)

qed

moreover have \( s = n2 \Rightarrow t = v' \Rightarrow ?\text{thesis} \)

using eulerian-split[|of G1 G2 n1 ps1 n1 v' (rev-path ps2) n2 w| merge-G1-G2]

by (metis \( \langle\text{edges} G1 \cap \text{edges} G2 = \{\}\rangle \langle n1 \neq n2 \rangle \langle\text{nodes} G1 \cap \text{nodes} G2 = \{\}\rangle \)

valid-unMultigraph G1) valid-unMultigraph G2) n1(1) n1(2) n2(1)

n2(2) ps1 s-ps2-t
valid-unMultigraph.eulerian-rev

moreover have \( s = v' \Rightarrow t = n2 \Rightarrow ?\text{thesis} \)

using eulerian-split[|of G1 G2 n1 ps1 n1 v' ps2 n2 w| merge-G1-G2]

by (metis \( \langle\text{edges} G1 \cap \text{edges} G2 = \{\}\rangle \langle n1 \neq n2 \rangle \langle\text{nodes} G1 \cap \text{nodes} G2 = \{\}\rangle \)

valid-unMultigraph G1) valid-unMultigraph G2) n1(1) n1(2) n2(1) n2(2)

ps1 s-ps2-t)

ultimately show \(?\text{thesis} by auto\)

qed

ultimately show \( \exists v \in \text{nodes} G. \exists v' \in \text{nodes} G. \exists ps. \text{odd} (\text{degree} v G) \land \text{odd} (\text{degree} v' G) \land v \neq v' \land \text{valid-unMultigraph.is-Eulerian-trail} G v ps v' \)

by auto

next
case less

assume finite (edges G) and finite (nodes G) and valid-unMultigraph G and nodes G \(\neq \{\}\)

and valid-unMultigraph.connected G and num-of-odd-nodes G = 0

show \( \forall v \in \text{nodes} G. \exists ps. \text{valid-unMultigraph.is-Eulerian-circuit} G v ps v \)

proof (rule cases card (nodes G) = 1)

fix v assume v \in nodes G

assume card (nodes G) = 1

hence nodes G = \{v\}

using \( \langle v \in \text{nodes} G \rangle \text{ card-Suc-eq[|of nodes G 0|] empty-iff insert-iff[|of - v|}\)

by auto

have edges G = \{\}

proof (rule ccontr)

assume edges G \(\neq \{\}\)

then obtain e1 e2 e3 where e: (e1, e2, e3) \in edges G by (metis ex-in-conv prod-cases3)

hence e1 = e3 using \( \langle\text{nodes} G = \{v\}\rangle \)

by (metis (hide-lams, no-types) append-nil valid-unMultigraph.is-trail-rev 69
val valid-unMultigraph.is-trail.simps(1) (valid-unMultigraph G) singletonE

val valid-unMultigraph.is-trail-split valid-unMultigraph.singleton-distinct-path
thus False by (metis e (valid-unMultigraph G) valid-unMultigraph.no-id)
qed

hence valid-unMultigraph.is-Eulerian-circuit G v [] v
by (metis (nodes G) = {v} insert-subset (valid-unMultigraph G) rem-unPath.simps(1))

subsetI valid-unMultigraph.is-trail.simps(1)
valid-unMultigraph.is-Eulerian-circuit-def
valid-unMultigraph.is-Eulerian-trail-def
thus \exists ps. valid-unMultigraph.is-Eulerian-circuit G v ps v by auto
next
fix v assume v \in \text{nodes} G
assume card (\text{nodes} G) \neq 1
moreover have card (\text{nodes} G) \neq 0 using (\text{nodes} G \neq \{\})
by (metis card-eq-0-iff (finite (\text{nodes} G)))
ultimately have card (\text{nodes} G) \geq 2 by auto
then obtain n where card (\text{nodes} G) = Suc (Suc n)
by (metis Nat.le-iff-add add-2-eq-Suc)
then obtain v̂ where (\text{nodes} G) \neq n by (auto dest: card-eq-SucD)

then obtain v' w where (v,w,v') \in \text{edges} G

proof —
  assume \text{pre}:\forall w' . (v,w,w') \in \text{edges} G \implies \text{thesis}
  assume \exists n \in \text{nodes} G . n \neq v
  then obtain ps where ps : v'. valid-graph.is-path G v ps v' \land ps \neq Nil
  using valid-unMultigraph-def
  by (metis (full-types) (v \in \text{nodes} G) (valid-unMultigraph G) valid-graph.is-path.simps(1))

then obtain v0 w v' where \text{ps}: Cons \langle v0,w,v' \rangle \text{ ps'} by (metis neq-Nil-conv prod-cases3)

hence v0 = v
using valid-unMultigraph-def
by (metis (valid-unMultigraph G) ps valid-graph.is-path.simps(2))

hence \langle v,w,v' \rangle \in \text{edges} G
using valid-unMultigraph-def
by (metis \exists ps'. ps = \langle v0, w, v' \rangle \# ps' (valid-unMultigraph G) ps
valid-graph.is-path.simps(2))

thus ?thesis by (metis pre)
qed

have all-even:\forall x \in \text{nodes} G . even (\text{degree} x G)
using (finite (\text{nodes} G)) (num-of-odd-nodes G = 0)
unfolding num-of-odd-nodes-def add-nodes-set-def by auto
have odd-v: odd (\text{degree} v (\text{del-unEdge} v w v' G))
using (v \in \text{nodes} G) all-even valid-unMultigraph.del-UnEdge-even[OF
(valid-unMultigraph G)
\langle v, w, v' \rangle \in \text{edges} G \land \text{finite} (\text{edges} G)]
unfolding odd-nodes-set-def by auto
have odd-

have all-even valid-graph.E-validD(2)[OF - ⟨(v, w, v') ∈ edges G] valid-unMultigraph G

have odd-nodes-set-def odd-nodes-set-def
by auto
have valid-unMulti:valid-unMultigraph (del-unEdge v w v' G)
by (metis del-unEdge-valid ⟨valid-unMultigraph G)
moreover have valid-graph: valid-graph (del-unEdge v w v' G)
using valid-unMultigraph-def del-undirected
by (metis valid-unMultigraph G; delete-edge-valid)
moreover have fin-E': finite(edges (del-unEdge v w v' G))
using (finite(edges G)); unfolding del-unEdge-def by auto
moreover have fin-V': finite(nodes (del-unEdge v w v' G))
using (finite(nodes G)); unfolding del-unEdge-def by auto
moreover have less-card:card(edges (del-unEdge v w v' G))<card(edges G)
unfolding del-unEdge-def using ⟨(v,w,v')∈edges G
by (metis Diff-insert2 card-Diff2-less ⟨finite (edges G)⟩ valid-unMultigraph G)
select-convs(2) valid-unMultigraph.corres
moreover have num-of-odd-nodes (del-unEdge v w v' G) = 2
using ⟨valid-unMultigraph G⟩ ⟨num-of-odd-nodes G = 0⟩ ⟨v ∈ nodes G⟩
all-even del-UnEdge-even-even[OF valid-unMultigraph G; ⟨finite (edges G), (finite (nodes G))⟩ ⟨(v, w, v') ∈ edges G] valid-graph.E-validD(2)[OF - ⟨(v, w, v') ∈ edges G]

unfolding valid-unMultigraph-def by auto
moreover have valid-unMultigraph.connected (del-unEdge v w v' G)
using (finite (edges G)); (finite (nodes G)); valid-unMultigraph G
valid-unMultigraph.connected G
by (metis ⟨(v, v, v') ∈ edges G; all-even valid-unMultigraph.del-unEdge-even-connectivity)
moreover have nodes(del-unEdge v w v' G) ≠ ∅
by (metis ⟨v ∈ nodes G; del-UnEdge-node emptyE)
ultimately obtain n1 n2 ps where
n1-n2:
n1 ∈ nodes (del-unEdge v w v' G)
n2 ∈ nodes (del-unEdge v w v' G)
odd (degree n1 (del-unEdge v w v' G))
odd (degree n2 (del-unEdge v w v' G))
n1 ≠ n2
and
ps-eulerian:
valid-unMultigraph.is-Eulerian-trail (del-unEdge v w v' G) n1 ps n2
by (metis ⟨num-of-odd-nodes (del-unEdge v w v' G) = 2⟩ less.hyps(1))
have \( n_1 = v \to n_2 = v' \to \text{valid-unMultigraph.is-Eulerian-circuit} G v (ps@[(v', w, v)]) \)

using ps-eulerian

by (metis \((v, w, v') \in \text{edges} G\) delete-edge-sym \(\text{valid-unMultigraph} G\)
  \(\text{valid-unMultigraph.corres} \text{valid-unMultigraph.eulerian-cons'}\)
  \(\text{valid-unMultigraph.is-Eulerian-circuit-def}\))

moreover have \( n_1 = v' \to n_2 = v' \to \exists ps. \text{valid-unMultigraph.is-Eulerian-circuit} G v ps v \)

by (metis \((v, w, v') \in \text{edges} G\) \(\text{valid-unMultigraph.eulerian-cons} \text{valid-unMultigraph.is-Eulerian-circuit-def}\))

moreover have \( (n_1 = v \land n_2 = v') \lor (n_2 = v \land n_1 = v') \)

by (metis \((v, w, v') \in \text{edges} G\) \(\text{finite} (\text{edges} G)\)
  \(\text{valid-unMultigraph} G\) \(n_1-n_2(1) n_1-n_2(2) n_1-n_2(3) n_1-n_2(4) n_1-n_2(5)\)
  \(\text{singletonE}\)
  \(\text{valid-unMultigraph.degree-frame}\))

ultimately show \( \exists ps. \text{valid-unMultigraph.is-Eulerian-circuit} G v ps v \) by auto

qed

end

theory FriendshipTheory

imports MoreGraph ~~/src/HOL/Number-Theory/Number-Theory

begin

10 Common steps

definition (in valid-unSimpGraph) non-adj :: \( v \Rightarrow v' \Rightarrow \text{bool} \) where

non-adj \( v \land v' \equiv \forall v' v. v \in V \land v' \in V \land v \neq v' \land \neg \text{adjacent} v v' \)

lemma (in valid-unSimpGraph) no-quad:

assumes \( \forall u. v \in V \to u \in V \to v \neq u \to \exists! n. \text{adjacent} v n \land \text{adjacent} u n \)

shows \( \neg (\exists v1 v2 v3 v4. v2 \neq v4 \land v1 \neq v3 \land \text{adjacent} v1 v2 \land \text{adjacent} v2 v3 \land \text{adjacent} v3 v4 \land \text{adjacent} v4 v1) \)

proof

assume \( \exists v1 v2 v3 v4. v2 \neq v4 \land v1 \neq v3 \land \text{adjacent} v1 v2 \land \text{adjacent} v2 v3 \land \text{adjacent} v3 v4 \land \text{adjacent} v4 v1 \)

then obtain \( v1 v2 v3 v4 \) where

\( v2 \neq v4 \land v1 \neq v3 \land \text{adjacent} v1 v2 \land \text{adjacent} v2 v3 \land \text{adjacent} v3 v4 \land \text{adjacent} v4 v1 \)

by auto

hence \( \exists! n. \text{adjacent} v1 n \land \text{adjacent} v3 n \) using assms[of \( v1 v3 \)] by auto

thus False

by (metis \( \text{adjacent} v1 v2 \) \(\text{adjacent} v2 v3\) \(\text{adjacent} v3 v4\) \(\text{adjacent} v4 v1\)
  \(v2 \neq v4\)
  \(\text{adjacent-sym}\))

qed
lemma even-card-set:
  assumes finite A and ∀x∈A. f x∈A ∧ f x≠ x ∧ f (f x)=x
  shows even(card A) using assms
proof (induct card A arbitrary:A rule:less-induct)
case less
  have A={}⇒ ?case by auto
moreover have A≠{}⇒ ?case
proof
  assume A≠{}
  then obtain x where x∈A by auto
  hence f x∈A ∧ f x≠ x by (metis less.prems(2))+
  obtain B where B=B=A−{x,f x} by auto
  hence finite B using (finite A) by auto
moreover have card B<card A using B (finite A)
  by (metis Diff-insert (f x∈A) card-Diff-less)
moreover have ∀x∈B. f x∈B ∧ f x≠ x ∧ f (f x)= x
proof
  fix y assume y∈B
  hence f y≠y and f (f y)=y by (metis less.prems(2))+
moreover have f y∈B
proof (rule ccontr)
  assume f y∉B
  have f y∈A by (metis (y∈A) less.prems(2))
  hence f y∈{x,f x} by (metis B Diff-I f y∉B)
  moreover have f y=x ⇒ False
  by (metis B Diff-I f y∉B f (f y)= y (y∈B) singleton-I)
  moreover have f y=f x⇒ False
  by (metis B Diff-I f x∈A) (y∈B) insertCI less.prems(2))
  ultimately show False by auto
  qed
ultimately show f y∈B ∧ f y≠ y ∧ f (f y)= y by auto
  qed
ultimately have even (card B) by (metis (full-types) less.hyps)
moreover have {x,f x}⊆A using (f x∈A) (x∈A) by auto
moreover have card {x,f x}= 2 using (f x≠ x) by auto
ultimately show ?case using B (finite A) card-mono [of A {x, f x}]
  by (simp add: card-Diff-subset)
  qed
ultimately show ?case by metis
  qed

lemma (in valid-unSimpGraph) even-degree:
  assumes friend-assm;\forall v u. v∈V ⇒ u∈V ⇒ v≠u ⇒ \exists! n. adjacent v n ∧ adjacent u n
  and finite E
  shows ∀v∈V. even(degree v G)
proof
fix v assume v∈V obtain f where f:f = (λn. (SOME v'. n∈V → n≠v → adjacent n v' ∧ adjacent v v')) by auto
  have \( \bigwedge n. n\in V \rightarrow n\neq v \rightarrow (\exists v'. \text{adjacent } n \ v' \land \text{adjacent } v \ v') \)
  proof (rule,rule)
    fix n assume n ∈ V n ≠ v
    hence \( \exists! v'. \text{adjacent } n \ v' \land \text{adjacent } v \ v' \)
    using friend-assm[of n v] unfolding non-adj-def by auto
    thus \( \exists v'. \text{adjacent } n \ v' \land \text{adjacent } v \ v' \) by auto
  qed
  hence f-ex;\( \bigwedge n. (\exists v'. n\in V \rightarrow n\neq v \rightarrow \text{adjacent } n \ v' \land \text{adjacent } v \ v') \) by auto
  have \( \forall x\in\{n. \text{adjacent } v \ n\}. f \in\{n. \text{adjacent } v \ n\} \land f x\neq x \land f (f x) = x \)
  proof (rule contr)
    assume f (f x)≠x
    have adjacent (f x) (f (f x))
      using someI-ex[OF f-ex,of x]
      by (metis adjacent-V(2) adjacent-no-loop calculation(1) f mem-Collect-eq)
    moreover have adjacent (f (f x)) v
      using someI-ex[OF f-ex,of f x] by (metis adjacent-V(1) adjacent-sym calculation f)
    also have adjacent x (f x)
      using someI-ex[OF f-ex,of x] by (metis adjacent V(2) adjacent-no-loop)
    moreover have v≠f x
      by (metis \( f \in\{n. \text{adjacent } v \ n\} \land f x \neq x \land f (f x) = x \) by auto
    qed
    moreover have finite \( \{n. \text{adjacent } v \ n\} \)
      using even-card-set[of \( \{n. \text{adjacent } v \ n\} \) f] by auto
    thus even(\( \text{degree } v \ G \)) by (metis assms(2) degree-adjacent)
  qed
lemma (in valid-unStimpGraph) degree-two-windmill:
  assumes friend-assm:\(\forall v \ u. \ v \in V \implies u \in V \implies v \neq u \implies \exists! \ n. \ \text{adjacent} \ v \ n \wedge \text{adjacent} \ u \ n\)
  and \(\text{finite} \ E \ \text{and} \ \text{card} \ V \geq 2\)
  shows (\(\exists v \in V. \ \text{degree} \ v \ G = 2\)) \iff (\(\exists v. \ \forall n \in V. \ n \neq v \implies \text{adjacent} \ v \ n\))
proof
  assume \(\exists v \in V. \ \text{degree} \ v \ G = 2\)
  then obtain \(v\) where \(\text{degree} \ v \ G = 2\) by auto
  hence \(\text{card} \ \{n. \ \text{adjacent} \ v \ n\} = 2\) using degree-adjacent[OF finite E,of v] by auto
  then obtain \(v1\) \(v2\) where \(\{n. \ \text{adjacent} \ v \ n\} = \{v1,v2\}\) and \(v1 \neq v2\)
proof
  obtain \(v1\) \(S\) where \(\{n. \ \text{adjacent} \ v \ n\} = \text{insert} \ v1 \ S\) and \(v1 \notin S\) and \(\text{card} \ S = 1\)
  using \(\langle \text{card} \ \{n. \ \text{adjacent} \ v \ n\} = 2\rangle\) card-Suc-eq[of \(\{n. \ \text{adjacent} \ v \ n\} \ 1\)] by auto
then obtain \(v2\) where \(S = \text{insert} \ v2 \ \emptyset\)
  using card-Suc-eq[of \(S \ 0\)] by auto
  hence \(\{n. \ \text{adjacent} \ v \ n\} = \{v1,v2\}\) and \(v1 \neq v2\)
  using \(\langle \text{card} \ \{n. \ \text{adjacent} \ v \ n\} = \text{insert} \ v1 \ S\ \{v1 \notin S\} \rangle\) by auto
  thus \(?thesis\) using that[of \(v1\) \(v2\)] by auto
  qed
have \(\text{adjacent} \ v1\) \(v2\)
  proof
  obtain \(n\) where \(\text{adjacent} \ v \ n \ \text{adjacent} \ v1\) \(n\) using friend-assm[of \(v\) \(v\) \(v1\)]
  by (metis (full-types) adjacent-V(2) adjacent-sym insertI1 mem-Collect-eq v1v2)
  hence \(n \in \{n. \ \text{adjacent} \ v \ n\}\) by auto
  moreover have \(n \neq v1\) by (metis \(\langle \text{adjacent} \ v1\) \(n\rangle\) adjacent-no-loop)
  ultimately have \(n = v2\) using v1v2 by auto
  thus \(?thesis\) by (metis \(\langle \text{adjacent} \ v1\) \(n\rangle\))
  qed
have \(v1v2\)-adj:\(\forall x \in V. \ x \in \{n. \ \text{adjacent} \ v1\) \(n\} \cup \{n. \ \text{adjacent} \ v2\) \(n\}\)
proof
  fix \(x\) assume \(x \in V\)
  have \(x = v \implies x \in \{n. \ \text{adjacent} \ v1\) \(n\} \cup \{n. \ \text{adjacent} \ v2\) \(n\}\)
  by (metis \(\text{Un-iff} \ \text{adjacent-sym} \ \text{insertI1} \ \text{mem-Collect-eq} \ v1v2\))
  moreover have \(x \neq v \implies x \in \{n. \ \text{adjacent} \ v1\) \(n\} \cup \{n. \ \text{adjacent} \ v2\) \(n\}\)
  proof
  assume \(x \neq v\)
  then obtain \(y\) where \(\text{adjacent} \ v \ y \ \text{adjacent} \ x\) \(y\)
  using friend-assm[of \(v\) \(x\)]
  by (metis \(\text{Collect-empty-eq} \ \langle x \in V\rangle\) adjacent-V(1) all-not-in-conv insertCI v1v2)
  hence \(y = v1\) \(\lor\) \(y = v2\) using v1v2 by auto
  thus \(x \in \{n. \ \text{adjacent} \ v1\) \(n\} \cup \{n. \ \text{adjacent} \ v2\) \(n\}\) using \(\langle \text{adjacent} \ x\) \(y\rangle\)
  by (metis \(\text{UnI1} \ \text{UnI2} \ \text{adjacent-sym} \ \text{mem-Collect-eq}\))
  qed
ultimately show \(x \in \{n. \ \text{adjacent} \ v1\) \(n\} \cup \{n. \ \text{adjacent} \ v2\) \(n\}\) by auto
have \{n. adjacent v1 n\} - \{v2, v\} = {} \implies \exists v. \forall n \in V. n \neq v \implies adjacent v n

proof (rule exI[of - v2], rule, rule)
  fix n assume v1-adj: \{n. adjacent v1 n\} - \{v2, v\} = {} and n \in V and n \neq v2
  have n \in \{n. adjacent v2 n\}
  proof (cases n = v)
    case True
    show ?thesis by (metis True adjacent-sym insertI1 insert-commute mem-Collect-eq v1v2)
  next
    case False
    have \(n \notin \{n. adjacent v2 n\}\) by (metis DiffI False \langle n \neq v2 \rangle empty-iff insert-iff v2-adj)
    thus ?thesis by (metis Un-iff \langle n \in V \rangle v1v2-adj)
  qed
  thus adjacent v2 n by auto
qed

moreover have \{n. adjacent v2 n\} - \{v1, v\} \neq {} \implies \exists v. \forall n \in V. n \neq v \implies adjacent v n

proof (rule exI[of - v1], rule, rule)
  fix n assume v2-adj: \{n. adjacent v2 n\} - \{v1, v\} = {} and n \in V and n \neq v1
  have n \in \{n. adjacent v1 n\}
  proof (cases n = v)
    case True
    show ?thesis by (metis True adjacent-sym insertI1 mem-Collect-eq v1v2)
  next
    case False
    have \(n \notin \{n. adjacent v1 n\}\) by (metis DiffI False \langle n \neq v1 \rangle empty-iff insert-iff v1-adj)
    thus ?thesis by (metis Un-iff \langle n \in V \rangle v1v2-adj)
  qed
  thus adjacent v1 n by auto
qed

moreover have \{n. adjacent v1 n\} - \{v2, v\} \neq {} \implies \{n. adjacent v2 n\} - \{v1, v\} \neq {}

\Longrightarrow False

proof
  assume \{n. adjacent v1 n\} - \{v2, v\} \neq {} \{n. adjacent v2 n\} - \{v1, v\} \neq {} \{}
  then obtain a b where a: a \in \{n. adjacent v1 n\} - \{v2, v\}
    and b: b \in \{n. adjacent v2 n\} - \{v1, v\}
    by auto
  have a = b \Longrightarrow False
  proof
    assume a = b
    have adjacent v1 a using a by auto
    moreover have adjacent a v2 using b \langle a = b \rangle adjacent-sym by auto
    moreover have a \neq v by (metis DiffD2 \langle a = b \rangle b doubleton-eq-iff insertI1)
  qed
moreover have \( \text{adjacent} \ v2 \ v \)
by (metis (full-types) adjacent-sym inf-sup-aci(5) insertII insert-is-Un
mem-Collect-eq
\( v1v2 \))

moreover have \( \text{adjacent} \ v \ v1 \) by (metis (full-types) insertII mem-Collect-eq
\( v1v2 \))
ultimately show False using no-quad\([\text{OF friend-assm}]\]
using \( \langle v1 \neq v2 \rangle \) by auto
qed
moreover have \( a \neq b \Rightarrow \) False
proof –
assume \( a \neq b \)
morerover have \( a \in V \) using \( a \) by (metis DiffD1 adjacent-V \( (2) \) mem-Collect-eq)
morerover have \( b \in V \) using \( b \) by (metis DiffD1 adjacent-V \( (2) \) mem-Collect-eq)
ultimately obtain \( c \) where \( \text{adjacent} \ a \ c \ \text{adjacent} \ b \ c \)
using friend-assm\([\text{of a b}]\) by auto
hence \( c \in \{n. \ \text{adjacent} \ v1 \ n\} \cup \{n. \ \text{adjacent} \ v2 \ n\} \)
by (metis (full-types) adjacent-V \( (2) \) v1v2-adj)
morerover have \( c \in \{n. \ \text{adjacent} \ v1 \ n\} \Rightarrow \) False
proof –
assume \( c \in \{n. \ \text{adjacent} \ v1 \ n\} \)
hence \( \text{adjacent} \ v1 \ c \) by auto
moreover have \( \text{adjacent} \ c \ b \) by (metis \( \langle \text{adjacent} \ a \ c \ \text{adjacent-sym} \rangle \))
morerover have \( \text{adjacent} \ b \ v2 \)
by (metis (full-types) Diff-iff adjacent-sym b mem-Collect-eq)
morerover have \( \text{adjacent} \ v2 \ v1 \) by (metis \( \langle \text{adjacent} \ a \ c \ \text{adjacent-sym} \rangle \))
morerover have \( c \neq v2 \)
proof (rule ccontr)
assume \( \neg c \neq v2 \)
hence \( c=v2 \) by auto
hence \( \text{adjacent} \ v2 \ a \) by (metis \( \langle \text{adjacent} \ a \ c \ \text{adjacent-sym} \rangle \))
morerover have \( \text{adjacent} \ v2 \ v \)
by (metis adjacent-sym insert-iff mem-Collect-eq v1v2)
morerover have \( \text{adjacent} \ v1 \ v \)
using adjacent-sym v1v2 by auto
moreover have \( \text{adjacent} \ v1 \ a \) by (metis (full-types) Diff-iff a
mem-Collect-eq)
ultimately have \( a=v \) using friend-assm \([\text{of} \ v1 \ v2] \)
by (metis \( \langle v1 \neq v2 \rangle \) adjacent-V \( (1) \) )
thus False using \( a \) by auto
qed
moreover have \( b \neq v1 \) by (metis DiffD2 b insertII)
ultimately show False using no-quad\([\text{OF friend-assm}]\) by auto
qed
moreover have \( c \in \{n. \ \text{adjacent} \ v2 \ n\} \Rightarrow \) False
proof –
assume \( c \in \{n. \ \text{adjacent} \ v2 \ n\} \)
hence \( \text{adjacent} \ c \ v2 \) by (metis adjacent-sym mem-Collect-eq)
morerover have \( \text{adjacent} \ a \ c \) using \( \langle \text{adjacent} \ a \ c \rangle \).

77
moreover have adjacent v1 a by (metis (full-types) Diff_iff a mem-Collect-eq)
moreover have adjacent v2 v1 by (metis ⟨adjacent v1 v2⟩ adjacent-sym)
moreover have c ≠ v1
proof (rule ccontr)
  assume ¬ c ≠ v1
  hence c = v1 by auto
  moreover have adjacent v1 v2 by (metis ⟨adjacent v1 v2⟩ adjacent-sym)
moreover have adjacent v2 v
  hence adjacent v1 b by (metis adjacent-sym insert_iff mem-Collect-eq v1v2)
moreover have adjacent v1 v
  using adjacent-sym v1v2 by auto
moreover have adjacent v2 b
  by (metis Diff_iff b mem-Collect-eq)
ultimately have b = v
  using friend-assm [of v1 v2]
  by (metis ⟨v1 ≠ v2⟩ adjacent-V (1))
thus False using b by auto
qed
moreover have a ≠ v2
  by (metis DiffD2 a insertI1)
ultimately show False using no-quad [OF friend-assm]
  by auto

ultimately show False by auto
qed
ultimately show False by auto
qed
ultimately show False by auto
next
assume ∃ v. ∀ n ∈ V. n ≠ v → adjacent v n
then obtain v where v: ∀ n ∈ V. n ≠ v → adjacent v n by auto
obtain v1 where v1 ∈ V v1 ≠ v
proof (cases v ∈ V)
case False
  have V ≠ {} using (2 ≤ card V) by auto
  then obtain v1 where v1 ∈ V by auto
  thus ?thesis using False that [of v1] by auto
next
case True
then obtain S where V = insert v S v ∉ S
  using mk-disjoint-insert [OF True] by auto
moreover have finite V using (2 ≤ card V)
  by (metis add-leE card-infinite not-one-le-zero numeral-Bit0 numeral-One)
ultimately have 1 ≤ card S
  using (2 ≤ card V) card.insert [of S v] finite-insert [of v S] by auto
hence S ≠ {} by auto
then obtain v1 where v1 ∈ S by auto
hence v1 ≠ v using ⟨v ∈ S⟩ by auto
thus thesis using that [of v1] ⟨v1 ∈ S⟩ ⟨V = insert v S⟩ by auto
qed
hence v ∈ V using v by (metis adjacent-V (1))
then obtain v2 where adjacent v1 v2 adjacent v v2 using friend-assm [of v v1]
by (metis \langle v1 \in V \rangle (v1 \neq v))
have degree v1 G \# 2 \implies False
proof –
  assume degree v1 G \# 2
  hence \card \{n. adjacent v1 n\} \# 2 by (metis assms(2) degree-adjacent)
  have \{v,v2\} \subseteq \{n. adjacent v1 n\}
    by (metis \langle adjacent v1 v2 \rangle \langle v1 \in V \rangle (v1 \neq v) adjacent-sym bot-least)
insert-subset
mem-Collect-eq v)
moreover have v \# v2 using (adjacent v v2) adjacent-no-loop by auto
hence card \{v,v2\} = 2 by auto
ultimately have card \{n. adjacent v1 n\} \geq 2
  using adjacent-finite[OF finite E, of v1] by (metis card-mono)
  hence card \{n. adjacent v1 n\} \geq 3 using \langle card \{n. adjacent v1 n\} \# 2 \rangle by auto
then obtain v3 where v3 \in \{n. adjacent v1 n\} and v3 \notin \{v,v2\}
  by (metis \langle card \{n. adjacent v1 n\} \# 2 \rangle subsetI subset-antisym)
  hence adjacent v1 v3 by auto
moreover have adjacent v3 v using v
  by (metis v3 \notin \{v,v2\} adjacent-V(2) adjacent-sym calculation insertCI)
moreover have adjacent v2 v1 using (adjacent v v2) .
moreover have adjacent v2 v1 using (adjacent v v2) adjacent-sym by auto
moreover have v1 \# v using v1 \# v .
moreover have v3 \# v2 by (metis v3 \notin \{v,v2\} insert-subset subset-insertI)
ultimately show False using no-quad[OF friend-assm] by auto
qed
thus \exists v \in V. degree v G = 2 using \langle v1 \in V \rangle by auto
qed

lemma (in valid-unSimpGraph) regular:
  assumes friend-assm:\forall u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. adjacent v n \land adjacent u n
  and finite E and finite V and \neg (\exists v \in V. degree v G = 2)
  shows \exists k. \forall v \in V. degree v G = k
proof –
  \{ fix v u assume non-adj v u \}
  obtain v-adj where v-adj:v-adj=\{n. adjacent v n\} by auto
  obtain u-adj where u-adj:u-adj=\{n. adjacent u n\} by auto
  obtain f where f:f = (\lambda n. (\forall v'. (\exists v'\prime. n \in V \implies n \neq u \implies adjacent n v' \land adjacent u v'\prime)) by auto
  have \forall u. n \in V \implies n \neq u \implies (\exists v'\prime. adjacent n v' \land adjacent u v'\prime)
    proof (rule,rule)
      fix n assume n \in V n \neq u
      hence \exists v'\prime. adjacent n v' \land adjacent u v'\prime
        using friend-assm[of n u] (non-adj v u) unfolding non-adj-def by auto
      thus \exists v'\prime. adjacent n v' \land adjacent u v'\prime by auto
    qed

79
hence $f$-ex: $\forall n. (\exists v'. n \in V \rightarrow n \neq u \rightarrow \text{adjacent } n \ v' \land \text{adjacent } u \ v')$ by auto

obtain $\text{v-adj } u$ where $\text{v-adj } u \colon v$-adj-def $\vdash f \colon v$-adj by auto
have finite $u$-adj using $u$-adj adjacent-finite[ OF $\text{finite } E$ ] by auto
have finite $v$-adj using $v$-adj adjacent-finite[ OF $\text{finite } E$ ] by auto
hence finite $v$-adj-u using $v$-adj-u adjacent-finite[ OF $\text{finite } E$ ] by auto
have $\text{inj-on } f \colon v$-adj unfolding $\text{inj-on-def}$
proof (rule ccontr)
  assume $\neg (\forall x \in v$-adj. $\forall y \in v$-adj. $f x = y \rightarrow x = y)$
  then obtain $x \ y$ where $x \in v$-adj $y \in v$-adj $f x = f y \ x \neq y$ by auto
have $\exists x \in V \ y \in v$-adj\ transform $\vdash (\text{metis } (x \in v$-adj) adjacent-V (2) mem-Collect-eq v$-adj)
moreover have $x \neq u$ by (metis (non-adj $v$ $u$) $(x \in v$-adj) mem-Collect-eq non-adj-def v$-adj)
ultimately have adjacent $y$ $(f \ y)$ using $\text{someI-ex[ OF } f$-ex[of $y$]] by (metis f)
  hence $x \neq y \land v \neq f x \land \text{adjacent } v \ x \land \text{adjacent } x (f \ x) \land \text{adjacent } (f \ x)$
  $y$
  $\land \text{adjacent } y \ v$
  using $(\exists x \in v$-adj) $(y \in v$-adj) $(f x = f y) (x \neq y) (\text{adjacent } x (f \ x)) v$-adj adjacent-sym $(f x \neq v)$
  by auto
thus False using no-quad[ OF friend-assm] by auto
qed
then have card $v$-adj $=$ card $v$-adj-u by (metis card-image v$-adj$)
moreover have $v$-adj-u $\subseteq u$-adj
proof
  fix $x$ assume $x \in v$-adj-u
  then obtain $y$ where $y \in v$-adj
      and $x = (\text{SOME } v'$. $y \in V \rightarrow y \neq u \rightarrow \text{adjacent } y \ v'$ $\land \text{adjacent } u \ v')$
      using $\text{f-image-def } v$-adj-u by auto
  hence $y \in V \rightarrow y \neq u \rightarrow \text{adjacent } y \ x \land \text{adjacent } u \ x$ using $\text{someI-ex[ OF } f$-ex[of $y$]]
  $\vdash y \in v$-adj $\rightarrow y \neq u$ $\rightarrow \text{adjacent } y \ x \land \text{adjacent } u \ x$ using $\text{someI-ex[ OF } f$-ex[of $y$]]
  by auto
  moreover have $y \in V \ y \neq u$ by (metis (non-adj $v$ $u$) adjacent-V (2) mem-Collect-eq v$-adj)
moreover have $y \neq u$ by (metis (non-adj $v$ $u$) $(y \in v$-adj) mem-Collect-eq non-adj-def v$-adj)
ultimately have adjacent $u \ x$ by auto
thus $x \in u$-adj unfolding $u$-adj by auto
qed
moreover have card $v$-adj $=$ degree $v$ $G$ using degree-adjacent[ OF $\text{finite } E$ ]
of $v$ $\vdash v$-adj by auto
moreover have card $u$-adj $=$ degree $u$ $G$ using degree-adjacent[ OF $\text{finite } E$ ],
of u] u-adj by auto
ultimately have degree v G ≤ degree u G using (finite u-adj)
by (metis (inj-on f u-adj) card-inj-on-le u-adj-u)

hence non-adj-degree; \( \forall u. \) non-adj v u \( \Rightarrow \) degree v G = degree u G
by (metis adjacent-sym antisym non-adj-def)

have \( \text{card } V = 3 \) \( \Rightarrow \) thesis

proof

assume \( \text{card } V = 3 \)
then obtain \( v_1 v_2 v_3 \) where \( V = \{v_1, v_2, v_3\} \) \( v_1 \neq v_2 \) \( v_2 \neq v_3 \) \( v_1 \neq v_3 \)

proof –

obtain \( v_1 S_1 \) where \( V S_1 = \text{insert } v_1 S_1 \) and \( v_1 \notin S_1 \) and \( \text{card } S_1 \) = 2

using card-Suc-eq[of V 2] \( \langle \text{card } V = 3 \rangle \) by auto
then obtain \( v_2 S_2 \) where \( S_1 S_2 = S_1 \) = insert \( v_2 S_2 \) and \( v_2 \notin S_2 \) and \( \text{card } S_2 = 1 \)
using card-Suc-eq[of S 1] by auto
then obtain \( v_3 \) where \( S_2 = \{v_3\} \)
using card-Suc-eq[of S 2] by auto
hence \( V = \{v_1, v_2, v_3\} \) using \( V S_1 S_2 = \text{by auto} \)
moreover have \( v_1 \neq v_2 \) \( v_2 \neq v_3 \) \( v_1 \neq v_3 \) using \( V S_1 S_2 \) \( \langle v_1 \notin S_1 \rangle \) \( \langle v_2 \notin S_2 \rangle \)

\( V = \{v_1, v_2, v_3\} \) by auto
ultimately show ?thesis using that by auto

qed

obtain \( n \) where \( \text{adjacent } v_1 n \) \( \text{adjacent } v_2 n \)
using friend-assm[of v1 v2] by (metis \( \langle V = \{v_1, v_2, v_3\} \rangle \) \( \langle v_1 \neq v_2 \rangle \) insertI1

insertI2)
moreover hence \( n = v_3 \)
using \( \langle V = \{v_1, v_2, v_3\} \rangle \) adjacent-V(2) adjacent-no-loop
by (metis (mono-tags) empty-iff insertE)

moreover obtain \( n' \) where \( \text{adjacent } v_2 n' \) \( \text{adjacent } v_3 n' \)
using friend-assm[of v2 v3] by (metis \( \langle V = \{v_1, v_2, v_3\} \rangle \) \( \langle v_2 \neq v_3 \rangle \) insertI1

insertI2)
moreover hence \( n' = v_1 \)
using \( \langle V = \{v_1, v_2, v_3\} \rangle \) adjacent-V(2) adjacent-no-loop
by (metis (mono-tags) empty-iff insertE)

ultimately have \( \text{adjacent } v_1 v_2 \) and \( \text{adjacent } v_2 v_3 \) and \( \text{adjacent } v_3 v_1 \)
using adjacent-sym by auto

have \( \text{degree } v_1 G = 2 \)

proof –

have \( v_2 \in \{n. \text{adjacent } v_1 n\} \) and \( v_3 \in \{n. \text{adjacent } v_1 n\} \) and \( v_1 \notin \{n. \text{adjacent } v_1 n\} \)

using \( \langle \text{adjacent } v_1 v_2 \rangle \) \( \langle \text{adjacent } v_3 v_1 \rangle \) adjacent-sym
by (auto,metis adjacent-no-loop)

hence \( \{n. \text{adjacent } v_1 n\} = \{v_2, v_3\} \) using \( \langle V = \{v_1, v_2, v_3\} \rangle \) by auto
thus ?thesis using degree-adjacent[OF finite E,of v1 \( \langle v_2 \neq v_3 \rangle \)] by auto

qed

moreover have \( \text{degree } v_2 G = 2 \)

proof –

have \( v_1 \in \{n. \text{adjacent } v_2 n\} \) and \( v_3 \in \{n. \text{adjacent } v_2 n\} \) and \( v_2 \notin \{n. \text{adjacent } v_2 n\} \)

81
adjacent v2 n} using (adjacent v1 v2) ⟨adjacent v2 v3⟩ adjacent-sym
by (auto, metis adjacent-no-loop)
hence \{ n. adjacent v2 n \} = \{ v1, v3 \} using (\{ V = \{ v1, v2, v3 \} \) \) by force
thus ?thesis using degree-adjacent[OF finite E, of v2] ⟨ v1 ≠ v3 ⟩ by auto

qed
moreover have degree v3 G = 2
proof −
have v1 ∈ \{ n. adjacent v3 n \} and v2 ∈ \{ n. adjacent v3 n \} and v3 ≠ \{ n. adjacent v3 n \}
using (adjacent v3 v1) ⟨ adjacent v2 v3 ⟩ adjacent-sym
by (auto, metis adjacent-no-loop)
hence \{ n. adjacent v3 n \} = \{ v1, v2 \} using (V = \{ v1, v2, v3 \} \) by force
thus ?thesis using degree-adjacent[OF finite E, of v3] ⟨ v1 ≠ v2 ⟩ by auto
qed
ultimately show ∀ v ∈ V. degree v G = 2 using (V = \{ v1, v2, v3 \}) by auto

qed
moreover have card V = 2 ⇒ False
proof −
assume card V = 2
obtain v1 v2 where V = \{ v1, v2 \} v1 ≠ v2
proof −
obtain v1 S1 where VS1 = \{ v1, v2 \} by auto
hence V = \{ v1, v2 \} using VS1 by auto
moreover have v1 ≠ v2 using (v1 ∈ S1) ⟨ S1 = \{ v2 \} ⟩ by auto
ultimately show ?thesis using that by auto
qed
then obtain v3 where adjacent v1 v3 adjacent v2 v3
using friend-assm[of v1 v2] by auto
hence v3 ≠ v2 and v3 ≠ v1 by (metis adjacent-no-loop)+
hence v3 ∉ V using (V = \{ v1, v2 \} \) by auto
thus False using ⟨ adjacent v1 v3 ⟩ by (metis (full-types) adjacent-V(2))
qed
moreover have card V = 1 ⇒ ?thesis
proof
assume card V = 1
then obtain v1 where V = \{ v1 \} using card-eq-SucD[of V 0] by auto
have E = \{ \}
proof (rule ccontr)
assume E ≠ \{ \}
then obtain x1 x2 x3 where x = (x1, x2, x3) ∈ E by auto
hence x1 = v1 and x3 = v1 using (V = \{ v1 \} \) E-validD by auto
thus False using no-id x by auto
qed

82
hence degree \( v_1 \) \( G = 0 \) unfolding degree-def by auto
thus \( \forall v \in V. \degree v G = 0 \) using \( \{V = \{v_1\}\} \) by auto
qed
moreover have \( \text{card } V = 0 \implies \text{thesis} \)
proof
assume \( \text{card } V = 0 \)
hence \( V = \{\} \) using \( \text{finite } V \) by auto
thus \text{thesis} by auto
qed
moreover have \( \text{card } V \geq 4 \implies \neg(\exists \ v \ u. \text{non-adj } v u) = \Rightarrow \text{False} \)
proof
assume \( \neg(\exists \ v \ u. \text{non-adj } v u) \)
\( \text{card } V \geq 4 \)
hence \( \text{finite } B \) unfolding non-adj-def by auto
obtain \( v_1 \ v_2 \ v_3 \ v_4 \) where \( v_1 \in V \ v_2 \in V \ v_3 \in V \ v_4 \in V \ v_1 \neq v_2 \ v_1 \neq v_3 \ v_1 \neq v_4 \ v_2 \neq v_3 \ v_2 \neq v_4 \ v_3 \neq v_4 \)
proof
obtain \( v_1 \ B_1 \) where \( V = \text{insert } v_1 \ B_1 \ v_1 \notin B_1 \) \text{card } B_1 \geq 3 \text{ finite } B_1
using \( \text{card } V \geq 4 \) card-le-Suc-iff \[ \text{OF } \langle \text{finite } V, \text{of } 3 \rangle \] by auto
then obtain \( v_2 \ B_2 \) where \( B_1 = \text{insert } v_2 \ B_2 \ v_2 \notin B_2 \) \text{card } B_2 \geq 2 \text{ finite } B_2
using card-le-Suc-iff \[ \text{of } B_1 2 \] by auto
then obtain \( v_3 \ B_3 \) where \( B_2 = \text{insert } v_3 \ B_3 \ v_3 \notin B_3 \) \text{card } B_3 \geq 1 \text{ finite } B_3
using card-le-Suc-iff \[ \text{of } B_2 1 \] by auto
then obtain \( v_4 \ B_4 \) where \( B_3 = \text{insert } v_4 \ B_4 \ v_4 \notin B_4 \)
using card-le-Suc-iff \[ \text{of } B_3 0 \] by auto
have \( v_1 \in V \) by \( \text{metis } \langle V = \text{insert } v_1 \ B_1, \text{insert-subset order-refl} \rangle \)
moreover have \( v_2 \in V \) by \( \text{metis } \langle B_1 = \text{insert } v_2 \ B_2, v_2 \notin B_2, \text{card } B_2 \geq 2 \rangle \)
subset-insertI
moreover have \( v_3 \in V \) by \( \text{metis } \langle B_1 = \text{insert } v_2 \ B_2, 2 = \text{insert } v_3 \ B_3, \langle V = \text{insert } v_1 \ B_1, \text{insert-iff} \rangle \rangle \)
moreover have \( v_4 \in V \) by \( \text{metis } \langle B_1 = \text{insert } v_2 \ B_2, 2 = \text{insert } v_3 \ B_3, 3 = \text{insert } v_4 \ B_4, \langle V = \text{insert } v_1 \ B_1, \text{insert-iff} \rangle \rangle \)
moreover have \( v_1 \neq v_2 \)
by \( \text{metis } \langle \text{full-types }, \text{B1 = insert v2 B2, v1 \notin B1, insertI1} \rangle \)
moreover have \( v_1 \neq v_3 \)
by \( \text{metis } \langle \text{full-types }, B1 = \text{insert } v_2 \ B_2, \langle v_1 \notin B_1, \text{insert-iff} \rangle \rangle \)
moreover have \( v_1 \neq v_4 \)
by \( \text{metis } \langle B_1 = \text{insert } v_2 \ B_2, B_2 = \text{insert } v_3 \ B_3, \langle v_1 \notin B_1, \text{insert-iff} \rangle \rangle \)
moreover have \( \langle v_1 \notin B_1, \text{insert-iff} \rangle \)
moreover have \( v_2 \neq v_3 \)
by \( \text{metis } \langle \text{full-types }, B_2 = \text{insert } v_3 \ B_3, v_2 \notin B_2, \text{insertI1} \rangle \)
moreover have \( v_2 \neq v_4 \)
by \( \text{metis } \langle B_2 = \text{insert } v_3 \ B_3, B_3 = \text{insert } v_4 \ B_4, \langle v_2 \notin B_2, \text{insert-iff} \rangle \rangle \)
moreover have \( v_3 \neq v_4 \)
ultimately show thesis using that by auto
qed

hence adjacent v1 v2 using non-non-adj by auto
moreover have adjacent v2 v3 using non-non-adj by (metis ⟨v2 ∈ V⟩ ⟨v2 ≠ v3⟩ ⟨v3 ∈ V⟩)
moreover have adjacent v3 v4 using non-non-adj by (metis ⟨v3 ∈ V⟩ ⟨v3 ≠ v4⟩ ⟨v4 ∈ V⟩)
moreover have adjacent v4 v1 using non-non-adj by (metis ⟨v1 ∈ V⟩ ⟨v1 ≠ v4⟩ ⟨v4 ∈ V⟩)
ultimately show False using no-quad[OF friend-assm]

by (metis ⟨v1 ≠ v3⟩ ⟨v2 ≠ v4⟩)

qed

moreover have card V ≥ 4 ⇒ (∃ u. non-adj v u) ⇒ thesis proof −
assume (∃ v u. non-adj v u) card V ≥ 4
then obtain w where non-adj v u by auto
then obtain w where adjacent v w and adjacent u w
and unique:∀ n. adjacent v n ∧ adjacent u n −→ n = w
using friend-assm[of v u] unfolding non-adj-def by auto
have ∀ n ∈ V. degree n G = degree v G
proof
fix n assume n ∈ V
moreover have n = v −→ degree n G = degree v G by auto
moreover have n = u −→ degree n G = degree v G
using non-adj-degree ⟨non-adj v w⟩ by auto
moreover have n ≠ v −→ n ≠ u −→ n ≠ w −→ degree n G = degree v G
proof −
assume n ≠ v n ≠ u n ≠ w
have non-adj v n −→ degree n G = degree v G by (metis non-adj-degree)
moreover have non-adj u n −→ degree n G = degree v G
by (metis ⟨non-adj v w⟩ non-adj-degree)
moreover have ¬ non-adj u n −→ ¬ non-adj v n −→ degree n G = degree v G
by (metis ⟨n ∈ V⟩ ⟨n ≠ w⟩ ⟨non-adj v w⟩ non-adj-def unique)
ultimately show degree n G = degree v G by auto
qed
moreover have n = w −→ degree n G = degree v G
proof −
assume n = w
moreover have ¬ (∃ v. ∀ n ∈ V. n ≠ v −→ adjacent v n)
using ⟨card V ≥ 4⟩ degree-two-windmill assms(2) assms(4) friend-assm
by auto
ultimately obtain w1 where w1 ∈ V w1 ≠ w non-adj w w1
by (metis ⟨n ∈ V⟩ ⟨n ≠ w⟩ ⟨non-adj v w⟩ non-adj-def)
have w1 = v −→ degree n G = degree v G
by (metis ⟨n = w⟩ ⟨non-adj v w1⟩ non-adj-degree)
moreover have w1 = u −→ degree n G = degree v G
by (metis ⟨adjacent u w⟩ ⟨non-adj w w1⟩ adjacent-sym non-adj-def)
moreover have \( w1 \neq u \Rightarrow w1 \neq v \Rightarrow \text{degree } n \ G = \text{degree } v \ G \)

by \((\text{metis } (n = w) \ (\text{non-adj } v \ w) \ (\text{non-adj } w \ w1) \ \text{non-adj-def})\)

ultimately show \( \text{degree } n \ G = \text{degree } v \ G \) by \text{auto}

qed

thus \(?\text{thesis}\) by \text{auto}

qed

ultimately show \(?\text{thesis}\) by \text{force}

qed

11 Exclusive steps for combinatorial proofs

fun \(\text{in } \text{valid-unSimpGraph}\) \(\text{adj-path} :: 'v \Rightarrow 'v \ \text{list} \Rightarrow \text{bool} \ \text{where}\)

\(\text{adj-path} \ v \ [] = (v \in V)\)
\| \(\text{adj-path} \ v \ (u \# \ us) = (\text{adjacent } v \ u \wedge \text{adj-path } u \ us)\)

lemma \(\text{in } \text{valid-unSimpGraph}\) \(\text{adj-path-butlast}:\)

\(\text{adj-path} \ v \ ps \Rightarrow \text{adj-path } v \ (\text{butlast } ps)\)
by \((\text{induct } ps \ \text{arbitrary:}v, \text{auto})\)

lemma \(\text{in } \text{valid-unSimpGraph}\) \(\text{adj-path-V}:\)

\(\text{adj-path} \ v \ ps \Rightarrow \text{set } ps \subseteq V\)
by \((\text{induct } ps \ \text{arbitrary:}v, \text{auto})\)

lemma \(\text{in } \text{valid-unSimpGraph}\) \(\text{adj-path-V}':\)

\(\text{adj-path} \ v \ ps \Rightarrow v \in V\)
by \((\text{induct } ps \ \text{arbitrary:}v, \text{auto})\)

lemma \(\text{in } \text{valid-unSimpGraph}\) \(\text{adj-path-app}:\)

\(\text{adj-path} \ v \ ps \Rightarrow ps \neq [] \Rightarrow \text{adjacent } (\text{last } ps) \ u \Rightarrow \text{adj-path } v \ (ps@[u])\)
proof \((\text{induct } ps \ \text{arbitrary:}v)\)

case Nil
thus \(?\text{case}\) by \text{auto}
next

case \((\text{Cons } x \ xs)\)
thus \(?\text{case}\) by \((\text{cases } xs, \text{auto})\)
qed

lemma \(\text{in } \text{valid-unSimpGraph}\) \(\text{adj-path-app}':\)

\(\text{adj-path} \ v \ (ps \ @ \ [q]) \Rightarrow ps \neq [] \Rightarrow \text{adjacent } (\text{last } ps) \ q\)
proof \((\text{induct } ps \ \text{arbitrary:}v)\)

case Nil
thus \(?\text{case}\) by \text{auto}
next

case \((\text{Cons } x \ xs)\)
thus \(?\text{case}\) by \((\text{cases } xs, \text{auto})\)


**lemma** card-partition':

**assumes** \( \forall v \in A. \text{card } \{ n. R v n \} = k > 0 \text{ finite } A \)

\[ \forall v1 v2, v1 \neq v2 \rightarrow \{ n. R v1 n \} \cap \{ n. R v2 n \} = \emptyset \]

**shows** \( \text{card } (\bigcup v \in A. \{ n. R v n \}) = k \ast \text{card } A \)

**proof** –

have \( \bigwedge C. C \in (\lambda x. \{ n. R x n \}) \quad A \implies \text{card } C = k \)

**proof** –

fix \( C \) assume \( C \in (\lambda x. \{ n. R x n \}) \quad A \)

show \( \text{card } C = k \) by (metis (mono-tags) \( C \in (\lambda x. \{ n. R x n \}) \quad A \) \( \text{assms}(1) \)

imageE)

**qed**

moreover have \( \bigwedge C1 C2. C1 \in (\lambda x. \{ n. R x n \}) \quad A \implies C2 \in (\lambda x. \{ n. R x n \}) \quad A \)

\[ \implies C1 \cap C2 = \emptyset \]

**proof** –

fix \( C1 C2 \) assume \( C1 \in (\lambda x. \{ n. R x n \}) \quad A \) \( C2 \in (\lambda x. \{ n. R x n \}) \quad A \)

\( C1 \neq C2 \)

obtain \( v1 \) where \( v1 \in A \quad C1 = \{ n. R v1 n \} \) by (metis \( C1 \in (\lambda x. \{ n. R x n \}) \quad A \) \( \text{imageE} \))

obtain \( v2 \) where \( v2 \in A \quad C2 = \{ n. R v2 n \} \) by (metis \( C2 \in (\lambda x. \{ n. R x n \}) \quad A \) \( \text{imageE} \))

have \( v1 \neq v2 \) by (metis \( C1 = \{ n. R v1 n \} \); \( C1 \neq C2 \); \( C2 = \{ n. R v2 n \} \))

thus \( C1 \cap C2 = \emptyset \) by (metis \( \text{assms}(1) \) \( \text{assms}(2) \) \( \text{card-eq-0-iff finite-UN-I less-nat-zero-code} \))

**moreover have** \( \bigwedge (\lambda x. \{ n. R x n \}) \quad A \) = (\( \bigcup x \in A. \{ n. R x n \} \)) by auto

**moreover have** finite \((\lambda x. \{ n. R x n \}) \quad A \) by (metis \( \text{assms}(3) \) \( \text{finite-imageE} \))

**moreover have** finite \((\bigcup (\lambda x. \{ n. R x n \}) \quad A \)) by (metis \( \text{assms}(1) \) \( \text{assms}(2) \) \( \text{card-empty inf idem less-le} \))

**proof** –

have inj-on \((\lambda x. \{ n. R x n \}) \quad A \) unfolding inj-on-def

using \( \forall v1 v2, v1 \neq v2 \rightarrow \{ n. R v1 n \} \cap \{ n. R v2 n \} = \emptyset \)

by (metis \( \text{assms}(1) \) \( \text{assms}(2) \) \( \text{card-empty inf idem less-le} \))

thus \( ?\text{thesis} \) by (metis \( \text{card-image} \))

**qed**

ultimately show \( ?\text{thesis} \) using card-partition[of \((\lambda x. \{ n. R x n \}) \quad A \)] by auto

**qed**

**lemma** (in valid-unSimpGraph) path-count:

**assumes** \( k \text{-adj} : \forall v. v \in V \implies \text{card } \{ n. \text{adjacent } v n \} = k \) and \( v \in V \) and finite \( V \) and \( k > 0 \)

**shows** \( \text{card } \{ \text{ps. length } ps = l \land \text{adj-path } v ps \} = k \ast l \)

**proof** (induct \( l \) rule:nat.induct)

**case** zero

have \( \{ \text{ps. length } ps = 0 \land \text{adj-path } v ps \} = [] \) using \( \langle v \in V \rangle \) by auto

86
thus \texttt{case by auto}

\textbf{next}

\begin{enumerate}
\item \texttt{case (Suc n)}
\item \texttt{obtain ext where ext: ext = (λps ps'. ps' ≠ [] ∧ (butlast ps' = ps) ∧ adj-path v ps') by auto}
\item \texttt{have ∀ps ∈ \{ps. length ps = n ∧ adj-path v ps\}. card \{ps'. ext ps ps'\} = k by auto}
\item \texttt{proof}
\item \texttt{fix ps assume ps ∈ \{ps. length ps = n ∧ adj-path v ps\}}
\item \texttt{hence adj-path v ps and length ps = n by auto}
\item \texttt{obtain qs where qs:qs = \{n. if ps = [] then adjacent v n else adjacent (last ps) n\} by auto}
\item \texttt{hence card qs = k by auto}
\item \texttt{proof}
\item \texttt{− have \texttt{∀ps ∈ \{ps. length ps = n ∧ adj-path v ps\}. card \{ps'. ext ps ps'\} = k by auto}}
\item \texttt{moreover have \texttt{∀xs. xs ∈ \{ps'. ext ps ps'\} ⇒ xs ∈ \{ps'. ext ps ps'\}}}
\item \texttt{proof}
\item \texttt{− have \texttt{λq. ps@[q]] by auto}}
\item \texttt{ultimately show ext ps xs using ext \{ps'. ext ps ps'\]. by auto}}
\item \texttt{next}
\item \texttt{case False}
\item \texttt{fix xs assume xs ∈ \{ps'. ext ps ps'\}
\item \texttt{proof}
\item \texttt{− have \texttt{λq. ps@[q]] by auto}}
\item \texttt{ultimately show ext ps xs using ext \{ps'. ext ps ps'\]. by auto}}
\item \texttt{case False}
\item \texttt{fix xs assume xs ∈ \{ps'. ext ps ps'\}
\item \texttt{proof}
\item \texttt{− have \texttt{λq. ps@[q]] by auto}}
\item \texttt{ultimately show ext ps xs using ext \{ps'. ext ps ps'\]. by auto}}
\end{enumerate}
hence \( qs = \{ n, \text{ adjacent v n} \} \) \textbf{using} \( qs \) \textbf{by} \textit{auto}

\textbf{fix} \( xs \) \textbf{assume} \( xs \in \{ ps', \text{ ext ps ps}' \} \)

\hence \( xs \neq [] \) \textbf{and} \( \text{butlast} \, xs = ps \) \textbf{and} \( \text{adj-path} \, v \, xs \) \textbf{using} \textit{ext} \textbf{by} \textit{auto}

\textbf{thus} \( xs \in \text{ app }' \, qs \)

\textbf{using} \( \text{True} \, \text{app} \, (qs = \{ n, \text{ adjacent v n} \}) \)

\textbf{by} \( \text{(metis \, app-path-butlast-last-id \, app-butlast-self-conv2)} \)

\textit{image-iff}

\( \text{mem-Collect-eq} \)

\textbf{next}

\textbf{case} \( \text{False} \)

\textbf{fix} \( xs \) \textbf{assume} \( xs \in \{ ps', \text{ ext ps ps}' \} \)

\hence \( xs \neq [] \) \textbf{and} \( \text{butlast} \, xs = ps \) \textbf{and} \( \text{adj-path} \, v \, xs \) \textbf{using} \textit{ext} \textbf{by} \textit{auto}

\textbf{then obtain} \( q \) \textbf{where} \( xs = ps @ [q] \) \textbf{by} \( \text{(metis \, append-butlast-last-id)} \)

\hence \( \text{adjacent} \, (\text{last} \, ps) \, q \) \textbf{using} \( (\text{adj-path} \, v \, xs) = \text{False} \) \textbf{adj-path-app'} \textbf{by} \textit{auto}

\textbf{thus} \( xs \in \text{ app }' \, qs \) \textbf{using} \( qs \)

\textbf{by} \( \text{(metis \, (lifting, \, full-types) \, False \, \text{app} \, \text{imageI})} \)

\textbf{qed}

\textbf{ultimately show} \( \text{thesis} \) \textbf{by} \textit{auto}

\textbf{qed}

\textbf{moreover have} \( \text{inj-on} \, \text{app} \, qs \) \textbf{using} \text{app unfolding} \text{inj-on-def} \textbf{by} \textit{auto}

\textbf{ultimately show} \( \text{card} \, \{ ps', \text{ ext ps ps}' \} = k \) \textbf{by} \( \text{(metis \, \text{card} \, qs = k \, \text{card-image})} \)

\textbf{qed}

\textbf{moreover have} \( \forall ps1 \, ps2. \, ps1 \neq ps2 \rightarrow \{ n, \text{ ext ps1 n} \} \cap \{ n, \text{ ext ps2 n} \} = \{\} \)

\textbf{using} \text{ext} \textbf{by} \textit{auto}

\textbf{moreover have} \( \text{finite} \, \{ ps, \text{ length ps = n} \land \text{adj-path v ps} \} \)

\textbf{using} \( \text{Suc} \, \text{hyps assms by} \, \text{(auto intro; \, card-ge-0-finite)} \)

\textbf{ultimately have} \( \text{card} \, \{ \bigcup \nu \in \{ ps, \text{ length ps = n} \land \text{adj-path v ps} \}, \{ n, \text{ ext v n} \} \} = k \) \textbf{by} \( \text{(card-partition[of} \, \text{ps, length ps = n} \land \text{adj-path v ps} \, \text{ext} \, k] \, k > 0)} \textbf{by} \textit{auto}

\textbf{moreover have} \( \{ ps, \text{ length ps = n+1} \land \text{adj-path v ps} \} \)

\( = \{ \bigcup ps \in \{ ps, \text{ length ps = n} \land \text{adj-path v ps} \}, \{ ps', \text{ ext ps ps}' \} \} \)

\textbf{proof} –

\textbf{have} \( \bigwedge xs. \, xs \in \{ ps, \text{ length ps = n + 1} \land \text{adj-path v ps} \} \implies \)

\( xs \in (\bigcup ps \in \{ ps, \text{ length ps = n} \land \text{adj-path v ps} \}, \{ ps', \text{ ext ps ps}' \}) \)

\textbf{proof} –

\textbf{fix} \( xs \) \textbf{assume} \( xs \in \{ ps, \text{ length ps = n + 1} \land \text{adj-path v ps} \} \)

\hence \( \text{length} \, xs = n + 1 \) \textbf{and} \( \text{adj-path} \, v \, x \) \textbf{by} \textit{auto}

\hence \( \text{butlast} \, xs \in \{ ps, \text{ length ps = n} \land \text{adj-path v ps} \} \)

\textbf{using} \( \text{adj-path-butlast length-butlast mem-Collect-eq by} \textit{auto} \)

\textbf{thus} \( xs \in (\bigcup ps \in \{ ps, \text{ length ps = n} \land \text{adj-path v ps} \}, \{ ps', \text{ ext ps ps}' \}) \)

\textbf{using} \( (\text{adj-path} \, v \, x \, \text{; length} \, xs = n + 1) = \text{UN-iff} \, \text{ext} \text{length-greater-0-conv} \)

\( \text{mem-Collect-eq} \)

\textbf{by} \textit{auto}

\textbf{qed}

\textbf{moreover have} \( \bigwedge xs. \, xs \in (\bigcup ps \in \{ ps, \text{ length ps = n} \land \text{adj-path v ps} \}, \{ ps'. \}) \)

88
\[
\text{ext } ps \text{ ps' } \implies \quad \begin{align*}
x \in \{ & \text{ps. length } ps = n + 1 \land \text{adj-path } v \text{ ps} \} \quad \\
\text{proof} & - \quad \begin{align*}
\text{fix } x & \text{ assume } x \in \bigcup \{ \text{ps. length } ps = n \land \text{adj-path } v \text{ ps} \}, \{ \text{ps'. ext } ps \text{ ps'} \} \\
\text{then obtain } y & \text{ where } \text{length } y = n \land \text{adj-path } v \text{ y ext } y \text{ xs by auto} \\
\text{hence } x & \in \{ \text{ps. length } ps = n + 1 \land \text{adj-path } v \text{ ps} \} \\
\text{by } & (\text{metis } (\text{lifting, full-types}) \langle \text{ext } y \text{ xs} \rangle) \text{ ext mem-Collect-eq} \\
\text{qed} & \\
\text{ultimately show } \text{thesis by fast} \\
\text{q} & \\
\text{ultimately show } \text{card } \{ \text{ps. length } ps = (\text{Suc } n) \land \text{adj-path } v \text{ ps} \} = k \land (\text{Suc } n) \\
\text{using } & \text{Suc.hyps by auto} \\
\text{q} & \\
\end{align*}
\]

**lemma** (in `valid-unSimpGraph`) `total-v-num`:
- assumes `friend-assm`:`\forall u \in V \implies u \in V \implies v \neq u \implies \exists! n. \text{adjacent } v \text{ n} \land \text{adjacent } u \text{ n}`
- and `finite E and finite V and V \neq \{\}` and `\forall v \in V. \text{degree } v G = k` and `k > 0`
- shows `\text{card } V = k^2 k - k + 1`

**proof** –
- have `k-adj`:`\forall v \in V \implies \text{card } \{ \text{n. adjacent } v \text{ n} \} = k` by `(metis \text{assms}(2) \text{assms}(5) \text{degree-adjacent})`
- obtain `v` where `v \in V` using `<V \neq \{\}>` by `auto`
- obtain `l2-eq-v` where `l2-eq-v`:`\{ \text{ps. length } ps = 2 \land \text{adj-path } v \text{ ps} \land \text{last } ps = v \}` by `auto`
- have `\text{card } l2-eq-v = k`

**proof** –
- obtain `hds` where `hds`:`hds = hds' l2-eq-v` by `auto`
- moreover have `hds = \{ \text{n. adjacent } v \text{ n} \}`

**proof** –
- have `\forall x. x \in hds \implies x \in \{ \text{n. adjacent } v \text{ n} \}`

**proof** –
- fix `x` assume `x \in hds`
- then obtain `ps` where `\text{hd } ps = x` length `ps = 2` adj-path `v ps` last `ps = v`
- using `hds l2-eq-v` by `auto`
- thus `\text{adjacent } v x` by `(metis \text{full-types}) \text{adj-path.simps}(2) \text{list.sel}(1) \text{length-0-conv}`

**neq-nil-conv**
- `\text{zero-neq-numeral})`

**qed**

moreover have `\forall x. x \in \{ \text{n. adjacent } v \text{ n} \} \implies x \in hds`

**proof** –
- fix `x` assume `x \in \{ \text{n. adjacent } v \text{ n} \}`
- obtain `ps` where `ps = [x, v]` by `auto`
- hence `\text{hd } ps = x` and `\text{length } ps = 2` and `\text{adj-path } v \text{ ps and last } ps = v`
- using `x \in \{ \text{n. adjacent } v \text{ n} \}, \text{adjacent-sym by auto}
- thus `x \in hds` by `(metis \text{lifting, mono-tags}) \text{hds image-eqI l2-eq-v}`

89
mem-Collect-eq)

qed
ultimately show hds={n. adjacent v n} by auto
qed
moreover have inj-on hd l2-eq-v unfolding inj-on-def
proof (rule+)
fix x y assume x ∈ l2-eq-v y ∈ l2-eq-v hd x = hd y
hence length x=2 and last x=last y and length y=2
using l2-eq-v by auto
hence x!1=y!1
using last-conv-nth[of x] last-conv-nth[of y] by force
moreover have x!0=y!0
using (hd x=hd y) (length x=2) (length y=2)
by (metis hd-conv-nth length-greater-0-conv)
ultimately show x=y using (length x=2) (length y=2)
using nth-equalityI[of x y]
by (metis One-nat-def less-2-cases)
qed
ultimately show card l2-eq-v=k using k-adj[OF v∈V] by (metis card-image)
qed
obtain l2-neq-v where l2-neq-v
using l2-neq-v l2-eq-v
by auto
have card l2-neq-v = k∗k−k
proof −
obtain l2-v where l2-v
using l2-v l2-neq-v
by auto
hence card l2-v=k∗k using path-count[OF k-adj,of v 2] ⟨0<k⟩ ⟨finite V⟩ ⟨v∈V⟩
by (simp add: power2-eq-square)
hence finite l2-v using ⟨k>0⟩ by (metis card-infinite mult-is-0 neq0-conv)
moreover have l2-v=l2-neq-v ∪ l2-eq-v using l2-v l2-neq-v l2-eq-v by auto
moreover have l2-neq-v ∩ l2-eq-v = {} using l2-neq-v l2-eq-v by auto
ultimately have card l2-neq-v = card l2-v − card l2-eq-v
by (metis Int-commute Nat.add-0-right Un-commute card-Diff-subset-Int card-Un-Int)
card-at-0-iff diff-add-inverse finite-Diff finite-Un inf-sup-absorb
less-nat-zero-code)
thus card l2-neq-v = k∗k−k using ⟨card l2-eq-v=k⟩ using ⟨card l2-v=k∗k⟩
by auto
qed
moreover have bij-betw last l2-neq-v {n. n∈V ∧ n≠v}
proof −
have last {l2-neq-v = {n. n∈V ∧ n≠v}
proof −
have {x. x∈ last {l2-neq-v = x∈{n. n∈V ∧ n≠v}
proof
fix x assume x∈last {l2-neq-v
then obtain ps where length ps = 2 adj-path v ps last ps=x last ps≠v
using l2-neq-v by auto
hence (last ps)∈V

90
by (metis (full-types) adj-path-V last-in-set length-0-conv set-rev-mp
zero-neq-numeral)
thus \( x \in V \land x \neq v \) using \(<last ps=x>\) \(<last ps\neq v>\) by auto
qed
moreover have \( \land x. x \in \{ n. n \in V \land n \neq v \} \Longrightarrow x \in \text{last' l2-neq-v} \)
proof –
fix \( x \) assume \( x \in \{ n. n \in V \land n \neq v \} \)
then obtain \( y \) where \( \text{adjacent v y adjacent x y} \)
using friend-assm[\( v \in V \)] by auto
hence \( \text{adj-path v [y,x]} \) using \( \text{adjacent-sym [of x y]} \) by auto
hence \( \langle x,y,x\rangle \text{ of l2-neq-v} \) using \( \text{l2-neq-v x} \) by auto
thus \( x \in \text{last' l2-neq-v} \) by (metis imageI last.simps not-Cons-self2)
qed
ultimately show \(?\text{thesis}\) by fast
qed
moreover have \( \text{inj-on last l2-neq-v unfolding inj-on-def} \)
proof (rule,rule,rule)
fix \( x,y \) assume \( x \in \text{l2-neq-v} \) \( y \in \text{l2-neq-v} \) \( x = \text{last y} \)
thus \( \text{length x=2 and adj-path v x and last x\neq v and length y=2 and adj-path v y} \)
and \( \text{last y\neq v} \)
using \( \text{l2-neq-v}\) by auto
obtain \( x1 \) \( x2 \) \( y1 \) \( y2 \) where \( x=x1[x1,x2] \) and \( y=y1[y1,y2] \)
proof –
\{
  fix \( l \) assume \( \text{length l=2} \)
  obtain \( h1 \) \( t \) where \( l=h1\#t \) and \( \text{length t=1} \)
  using \( \langle \text{length l=2} \rangle \) \( \text{Suc-length-conv [of 1 l]} \) by auto
  then obtain \( h2 \) where \( t=[h2] \)
  using \( \text{Suc-length-conv [of 0 l]} \) by auto
  have \( \exists h1 h2. l=[h1,h2] \) using \( \langle l=h1\#t \rangle \) \( \langle t=[h2] \rangle \) by auto \}
thus \(?\text{thesis}\) using \( \langle \text{length x=2} \rangle \) \( \langle \text{length y=2} \rangle \) by metis
qed
hence \( x2\neq v \) and \( y2\neq v \) using \( \langle \text{last x\neq v} \rangle \) \( \langle \text{last y\neq v} \rangle \) by auto
moreover have \( \text{adjacent v x1 and adjacent x2 x1 and x2\in V} \)
using \( \text{adj-path v x}\) \( \text{x adjacent-sym by auto} \)
multiper have \( \text{adjacent v y1 and adjacent y2 y1 and y2\in V} \)
using \( \text{adj-path v y}\) \( \text{y adjacent-sym by auto} \)
ultimately have \( \text{x1=y1 using friend-assm [v\in V]} \)
by (metis \( \langle \text{last x = last y}\rangle \) last-ConsL last-ConsR not-Cons-self2 \( x\ y \))
thus \( x=y \) using \( \text{x y}\) \( \langle \text{last x = last y}\rangle \) by auto
qed
ultimately show \(?\text{thesis}\) unfolding bij-betw-def by auto
qed
hence \( \text{card l2-neq-v} = \text{card} \{ n. n \in V \land n \neq v \} \) by (metis bij-betw-same-card)
ultimately have \( \text{card} \{ n. n \in V \land n \neq v\} = k*\text{k} - k \) by auto
moreover have \( \text{card V} = \text{card} \{ n. n \in V \land n \neq v\} + \text{card} \{ v\} \)
proof –
have \( V=\{ n. n \in V \land n \neq v\} \cup \{ v\} \) using \( \langle v\in V \rangle \) by auto
moreover have \( \{ n. n \in V \land n \neq v\} \cap \{ v\} = \{} \) by auto
ultimately show \( \text{thesis} \)

using \( \langle \text{finite } V \rangle \) card-Un-disjoint\( \{ n \in V . n \neq v \} \) finite-Un
by auto

qed
ultimately show \( \text{card } V = k^2 - k + 1 \) by auto
qed

lemma rotate-eq: rotate1 \( xs = \text{rotate1 ys} \) \( \Rightarrow \) \( xs = ys \)
proof (induct \( xs \) arbitrary:ys)
  case Nil
  thus \( \text{?case by (metis rotate1-is-Nil-conv)} \)
next
  case (Cons \( n \) ns)
  hence \( ys \neq [] \) by (metis list.distinct(1) rotate1-is-Nil-conv)
  thus \( \text{?case using Cons by (metis butlast-snoc last-snoc list.exhaust rotate1.simps(2))} \)
qed

lemma rotate-diff: rotate \( m \) \( xs = \text{rotate } n \) \( xs \) \( \Rightarrow \) \( \text{rotate} (m - n) \) \( xs = xs \)
proof (induct \( m \) arbitrary:n)
  case 0
  thus \( \text{?case by auto} \)
next
  case (Suc \( m' \))
  hence \( n = 0 \) \( \Rightarrow \) \( \text{?case by auto} \)
  moreover have \( n \neq 0 \) \( \Rightarrow ?\text{case} \)
  proof -
    assume \( n \neq 0 \)
    then obtain \( n' \) where \( n = \text{Suc } n' \) by (metis nat.exhaust)
    hence \( \text{rotate } (m' - n') \) \( xs = xs \) by auto
    using \( \langle \text{rotate} (\text{Suc } m') \text{ xs = rotate n xs} \rangle \) rotate-eq rotate-Suc
    by auto
    hence \( \text{rotate} (m' - n') \) \( xs = xs \) by (metis Suc.hyps)
    moreover have \( \text{Suc } m' - n = m' - n' \)
      by (metis \( n' \) diff-Suc-Suc)
    ultimately show \( \text{?case by auto} \)
  qed
ultimately show \( \text{?case by fast} \)
qed

lemma (in valid-unSimpGraph) exist-degree-two:
assumes friend-assm: \( \exists v u . v \in V \quad \Rightarrow \quad u \in V \quad \Rightarrow \quad v \neq u \quad \Rightarrow \quad \exists ! n . \text{adjacent } v n \land \text{adjacent } u n \)
and finite \( E \) and finite \( V \) and card \( V \geq 2 \)
shows \( \exists v \in V . \text{degree } v G = 2 \)
proof (rule ccontr)
  assume \( \neg (\exists v \in V . \text{degree } v G = 2) \)
  hence \( \forall v . v \in V \Rightarrow \text{degree } v G \neq 2 \) by auto
  obtain \( k \) where \( k\text{-adj} : \forall v . v \in V \Rightarrow \text{card } \{ n . \text{adjacent } v n \} = k \) using regular[OF
have $k \geq 4$

proof
  obtain $v_1 \ v_2$ where $v_1 \in V \ v_2 \in V \ v_1 \neq v_2$
    using (card $V \geq 2$) by (metis ($\neg \exists v \in V. \ \text{degree} \ v \ G = 2$) assms(2))

moreover have $k \neq 0$
  proof
    assume $k = 0$
    obtain $v_3$ where adjacent $v_1 \ v_3$
      using friend-assm [OF $\langle v_1 \in V \rangle \ \langle v_2 \in V \rangle \ \langle v_1 \neq v_2 \rangle$]
      by auto
    hence $\text{card} \ \{n. \ \text{adjacent} \ v_1 \ n\} \neq 0$
      using adjacent-finite [OF $\langle \text{finite} \ E \rangle$]
      by auto
  qed

ultimately have $k \neq 0$ and $k \neq 1$ and $k \neq 3$ by auto

moreover have $k \neq 2$ using ($\And v. \ v \in V \Rightarrow \text{degree} \ v \ G \neq 2$) degree-adjacent

k-adj
  by (metis ($\exists v \in V. \ \text{degree} \ v \ G = 2$) assms(2))

ultimately show $?\text{thesis}$ by auto

obtain $T$ where $T = (\lambda l::\text{nat}. \ \{ps. \ \text{length} \ ps = l + 1 \ \wedge \text{adj-path} \ (hd \ ps) \ (tl \ ps)\} )$
  by auto

have $T\text{-count} : \And l::\text{nat}. \ \text{card} \ (T \ l) = (k + k - k + 1) * k^l$ using card-partition'

proof
  fix $l::\text{nat}$
  obtain $\text{ext} \ where \ \text{ext}\text{-ext} = (\lambda v \ ps. \ \text{adj-path} \ v \ (tl \ ps) \ \wedge \ hd \ ps = v \ \wedge \ \text{length} \ ps = l + 1)$
    by auto
  have $\forall v \in V. \ \text{card} \ \{ps. \ \text{ext} \ v \ ps\} = k^l$
    proof
      fix $v$ assume $v \in V$
      have $\And ps. \ ps \in tl \ \{ps. \ \text{ext} \ v \ ps\} \Rightarrow \ ps \in \{ps. \ \text{length} \ ps = l \ \wedge \ \text{adj-path} \ v \ ps\}$
        proof
          fix $ps$ assume $ps \in tl \ \{ps. \ \text{ext} \ v \ ps\}$
          then obtain $ps' \ where \ \text{adj-path} \ v \ (tl \ ps') \ hd \ ps' = v \ \text{length} \ ps' = l + 1$
            proof
              using $\text{ext}$ by auto
              hence $\text{adj-path} \ v \ ps$ and $\text{length} \ ps = l$ by auto
              thus $ps \in \{ps. \ \text{length} \ ps = l \ \wedge \ \text{adj-path} \ v \ ps\}$ by auto
            qed
          qed
        qed
    qed

ultimately have $\And ps. \ ps \in \{ps. \ \text{length} \ ps = l \ \wedge \ \text{adj-path} \ v \ ps\} \Rightarrow \ ps \in tl \ \{ps. \ \text{ext} \ v \ ps\}$
  proof
  qed
fix \( ps \) assume \( ps \in \{ ps. \ length \ ps = l \land \ adj\text{-}path \ v \ ps \} \)

hence \( length \ ps = l \land \ adj\text{-}path \ v \ ps \) by auto

moreover obtain \( ps' \) where \( ps' = v \# ps \) by auto

ultimately have \( adj\text{-}path \ v \ (tl \ ps') \) and \( hd \ ps' = v \) and \( length \ ps' = l + 1 \)

by auto

thus \( ps \in tl ' \{ ps. \ ext \ v \ ps \} \)

by (metis \( ps' = v \# ps \); ext image1 mem-Collect.eq list.sel(3))

qed

ultimately have \( tl ' \{ ps. \ ext \ v \ ps \} = \{ ps. \ length \ ps = l \land \ adj\text{-}path \ v \ ps \} \)

by fast

moreover have \( inj\text{-}on tl \{ ps. \ ext \ v \ ps \} \) unfolding inj-on-def

proof (rule,rule,rule)

fix \( x \ y \) assume \( x \in \text{Collect} \ (ext \ v) \ y \in \text{Collect} \ (ext \ v) \ tl \ x = tl \ y \)

hence \( hd \ x = hd \ y \land x \neq [] \land y \neq [] \) using ext by auto

thus \( x = y \) using \( (tl \ x = tl \ y) \) by (metis list.sel(1,3) list.exhaust)

qed

moreover have \( card \{ ps. \ length \ ps = l \land \ adj\text{-}path \ v \ ps \} = k \cdot l \)

using path-count[OF k-adj.of v \( l \) \( \langle 4 \leq k \rangle \) \( v \in V \) assms(3)]

by auto

ultimately show \( card \{ ps. \ ext \ v \ ps \} = k \cdot l \) by (metis card-image)

qed

moreover have \( \forall v l \ v2. v l \neq v2 \implies \{ n. \ ext \ v l \ n \} \cap \{ n. \ ext \ v2 \ n \} = \{ \} \)

using ext by auto

moreover have \( (\bigcup v \in V. \{ n. \ ext \ v \ n \}) = T l \)

proof –

have \( \forall ps. ps \in (\bigcup v \in V. \{ n. \ ext \ v \ n \}) \implies ps \in T l \) using T

proof –

fix \( ps \) assume \( ps \in (\bigcup v \in V. \{ n. \ ext \ v \ n \}) \)

then obtain \( v \) where \( v \in V \) adj-path \( v \) (tl \ ps) hd \ ps = v \ length \ ps = l \)

+ 1

using ext by auto

hence \( length \ ps = l + 1 \) and \( adj\text{-}path \ (hd \ ps) \ (tl \ ps) \) by auto

thus \( ps \in T l \) using T by auto

qed

moreover have \( \forall ps. ps \in T l \implies ps \in (\bigcup v \in V. \{ n. \ ext \ v \ n \}) \)

proof –

fix \( ps \) assume \( ps \in T l \)

hence \( length \ ps = l + 1 \) and \( adj\text{-}path \ (hd \ ps) \ (tl \ ps) \) using T by auto

moreover then obtain \( v \) where \( v = hd \ ps \) \( v \in V \)

by (metis adj-path.simps(1) adj-path.simps(2) adjacent-V(1) list.exhaust)

ultimately show \( ps \in (\bigcup v \in V. \{ n. \ ext \ v \ n \}) \) using ext by auto

qed

ultimately show \( ?thesis \) by auto

qed

ultimately have \( card \ (T l) = card \ V \ast k \cdot l \)

using card-partition[of \( V \) ext \( k \cdot l \) \( \langle 4 \leq k \rangle \) assms(3)] mult.commute nat-one-le-power

94
by auto
moreover have \( \text{card } V = (k \cdot k - k + 1) \)
  using \( \text{total-v-num[of friend-assm,of } k \text{]} \text{ k-adj degree-adjacent } \langle \text{finite } E \rangle \)
  \( \langle \text{finite } V \rangle \) \( \langle \text{card } V \geq 2 \rangle \) \( \langle 4 \leq k \rangle \text{ card-gt-0-iff} \)
  by force
ultimately show \( \text{card } (T \cdot l) = (k \cdot k - k + 1) \cdot k \cdot l \) by auto
qed
obtain \( C \) where \( C \cdot C = \langle \lambda l::\text{nat}. \{ \text{ps. length } ps = l+1 \land \text{adj-path } (hd \cdot ps) (tl \cdot ps) \} \rangle \)
  \( \land \) adjacent \( \langle \text{last } ps \rangle (hd \cdot ps)) \) by auto
obtain \( C \cdot \text{star} \) where \( C \cdot \text{star}: C \cdot \text{star} = \langle \lambda l::\text{nat}. \{ \text{ps. length } ps = l+1 \land \text{adj-path } (hd \cdot ps) (tl \cdot ps) \} \rangle \)
  \( \land \) \( \langle \text{last } ps \rangle = (hd \cdot ps)) \) by auto
have \( \langle \lambda l::\text{nat}. \text{card } (C \cdot (l\cdot+1)) = k \cdot \text{card } (C \cdot \text{star } l) + \text{card } (T \cdot l - C \cdot \text{star } l) \rangle \)
proof
  fix \( l::\text{nat} \)
  have \( C \cdot (l\cdot+1) = \{ \text{ps. length } ps = l\cdot+2 \land \text{adj-path } (hd \cdot ps) (tl \cdot ps) \land \text{adjacent } \langle \text{last } ps \rangle (hd \cdot ps) \land \text{last } (butlast \cdot ps) = hd \cdot ps \} \cup \{ \text{ps. length } ps = l\cdot+2 \land \text{adj-path } (hd \cdot ps) (tl \cdot ps) \land \text{adjacent } \langle \text{last } ps \rangle (hd \cdot ps) \land \text{last } (butlast \cdot ps) = hd \cdot ps \} \) \( \land \) \( \text{adjacent } \langle \text{last } ps \rangle (hd \cdot ps) \land \text{last } (butlast \cdot ps) \not= hd \cdot ps \) using \( C \) by auto
  moreover have \( \{ \text{ps. length } ps = l\cdot+2 \land \text{adj-path } (hd \cdot ps) (tl \cdot ps) \land \text{adjacent } \langle \text{last } ps \rangle (hd \cdot ps) \land \text{last } (butlast \cdot ps) \not= hd \cdot ps \} = \{ \} \) by auto
  moreover have finite \( (C \cdot (l\cdot+1)) \)
  proof
    have \( C \cdot (l\cdot+1) \subseteq T \cdot (l\cdot+1) \) using \( C \cdot T \) by auto
    moreover have \( \langle k \cdot k - k + 1 \rangle \cdot k \cdot l \neq 0 \) using \( k \geq 4 \) by auto
    hence finite \( (T \cdot (l\cdot+1)) \) using \( T \cdot \text{count}[\text{of } l\cdot+1] \) by (metis card-infinite)
    ultimately show ?thesis by (metis finite-subset)
  qed
ultimately have \( \text{card } (C \cdot (l\cdot+1)) = \text{card } \{ \text{ps. length } ps = l\cdot+2 \land \text{adj-path } (hd \cdot ps) (tl \cdot ps) \} \land \text{adjacent } \langle \text{last } ps \rangle (hd \cdot ps) \land \text{last } (butlast \cdot ps) = hd \cdot ps \} \) \( \land \) \( \text{adjacent } \langle \text{last } ps \rangle (hd \cdot ps) \land \text{last } (butlast \cdot ps) \not= hd \cdot ps \) using \( C \cdot \text{Un-disjoint}[\text{of } \{ \text{ps. length } ps = l\cdot+2 \land \text{adj-path } (hd \cdot ps) (tl \cdot ps) \} \land \text{adjacent } \langle \text{last } ps \rangle (hd \cdot ps) \land \text{last } (butlast \cdot ps) = hd \cdot ps \} = k \cdot \text{card } (C \cdot \text{star } l) \)
proof

obtain ext where ext : ext = (λps ps'. ps' ≠ [] ∧ (butlast ps' = ps) ∧ adj-path (hd ps') (tl ps')) by auto

have ∀ ps ∈ (C-star l). card {ps'. ext ps ps'} = k

proof

fix ps assume ps ∈ C-star l

hence length ps = l + 1 and adj-path (hd ps) (tl ps) and last ps = hd ps

using C-star by auto

obtain qs where qs :qs = {v. adjacent (last ps) v} by auto

obtain app where app : app = (λv. ps @ [v]) by auto

have app' qs = {ps'. ext ps ps'}

proof

have ⋀ x. x ∈ app' qs ⇒ x ∈ {ps'. ext ps ps'}

proof

fix x assume x ∈ app' qs

then obtain y where adjacent (last ps) y = x ∈ ps @ [y] using qs

app by auto

moreover hence adj-path (hd x) (tl x)

by (cases tl ps = [], metis adj-path.simps(1) adj-path.simps(2) adjacent-V(2) append-Nil list.sel(1,3) hd-append snoc-eq-iff-butlast tl-append2, metis ↑adj-path (hd ps) (tl ps) adj-path-app hd-append)

ultimately show ext ps x using ext by (metis snoc-eq-iff-butlast) qed

moreover have ⋀ x. x ∈ {ps'. ext ps ps'} =⇒ x ∈ app' qs

proof

fix x assume x ∈ {ps'. ext ps ps'}

hence x ≠ [] and butlast x = ps and adj-path (hd x) (tl x)

using ext by auto

have adjacent (last ps) (last x)

proof (cases length ps = 1)

case True

hence length x = 2 using ↑butlast x = ps by auto

then obtain x1 t1 where x = x1 # t1 and length t1 = 1

using Suc-length-conv[of 1 x] by auto

then obtain x2 where t1 = [x2]

using Suc-length-conv[of 0 t1] by auto

have x = [x1, x2] using ↑x = x1 # t1, ↑t1 = [x2] by auto

thus adjacent (last ps) (last x)

using ↑adj-path (hd x) (tl x), ↑butlast x = ps by auto

next

case False

hence tl ps ≠ []

by (metis ↑length ps = l + 1) add-0-iff add-cancel-left' length-0-comp length-tl add.commute

moreover have adj-path (hd x) (tl ps ⊕ [last x])
ultimately have adjacent (last (tl ps)) (last x)
by auto
thus adjacent (last ps) (last x) by (metis (tl ps ≠ []) last-tl)
qed
thus x ∈ app ' qs using app qs
by (metis ⟨butlast x = ps x ≠ []⟩ append-butlast-last-id
mem-Collect-eq rev-image-eqI)
}
tl-append2
}

ultimately show ?thesis by auto
qed
moreover have inj-on app qs using app unfolding inj-on-def by auto
moreover have last ps∈ V
using (length ps = l + 1) ⟨adj-path (hd ps) (tl ps)⟩ adj-path-V
by (metis (last ps = hd ps) adj-path.simps(1) last-in-set last-tl
subset-code(1))

ultimately show card {ps'. ext ps ps'} = k by (metis card-image)

ultimately show card {ps'. ext ps ps'} = k
ultimately show ?thesis by (metis finite

moreover have finite (C-star l)
proof –
moreover have last ps∈ V
using length ps = l + 1 ⟨adj-path (hd ps) (tl ps)⟩ adj-path-V
by (metis (last ps = hd ps) adj-path.simps(1) last-in-set last-tl
subset-code(1))

ultimately show ?thesis by (metis finite-subset)
qed

moreover have ∀ ps1 ps2. ps1 ≠ ps2 → {ps’. ext ps ps’} ∩ {ps’. ext
ps2 ps’} = {}
using ext by auto

moreover have (⋃ ps∈ (C-star l). {ps’. ext ps ps’}) = {ps. length ps = l+2
∧ adj-path (hd ps) (tl ps) ∧ adjacent (last ps) (hd ps) ∧ last (butlast
ps)=hd ps}
proof –
moreover have (⋃ ps∈ (C-star l). {ps’. ext ps ps’}) = {ps. length ps =
l+2 ∧ adj-path (hd ps) (tl ps) ∧ adjacent (last ps) (hd ps) ∧ last (butlast
ps)=hd ps}
proof

fix x assume x ∈ (⋃ ps∈ C-star l. {ps’. ext ps ps’})
then obtain ps where ps∈ C-star l ext ps x by auto
hence length ps = l + 1 and adj-path (hd ps) (tl ps) and last ps = hd ps
and x ≠ [] and butlast x = ps adj-path (hd x) (tl x)
using C-star ext by auto
have length x = l + 2
  using ⟨butlast x = ps⟩ ⟨length ps = l + 1⟩ length-butlast by auto
moreover have adj-path (hd x) (tl x) by (metis adj-path (hd x)
(tl x));
moreover have adjacent (last x) (hd x)
proof
  have length x ≥ 2 using ⟨length x = l + 2⟩ by auto
  hence adjacent (last (butlast x)) (last x) using ⟨adj-path (hd x)
(tl x));
  hence adjacent (last x) (hd x) using ⟨butlast x = ps⟩ ⟨length ps = l + 1⟩
  by (cases x) auto
  thus ?thesis using adjacent-sym by auto
qed
moreover have last (butlast x) = hd x
by (metis ⟨butlast x = ps⟩ ⟨last ps = hd ps⟩ ⟨x ≠ []⟩ adjacent-no-loop
  butlast.simps(2) calculation(3) list.sel(1) last-Cons neq-Nil-conv)
ultimately show length x = l + 2 ∧ adj-path (hd x) (tl x)
  ∧ adjacent (last x) (hd x) ∧ last (butlast x) = hd x
  by auto
qed
moreover have \( \forall x. x \in \{ps. \text{length } ps = l + 2 ∧ \text{adj-path } (hd ps) (tl ps)
\} \), \( x \in (\bigcup \{ps \in (C-star l). \{ps'. \text{ext ps ps'}\}\}) \)
proof –
fix x assume \( x \in \{ps. \text{length } ps = l + 2 ∧ \text{adj-path } (hd ps) (tl ps)
\} \), \( x \in (\bigcup \{ps \in (C-star l). \{ps'. \text{ext ps ps'}\}\}) \)
hence length x = l + 2 and adj-path (hd x) (tl x) and adjacent (last x) (hd x)
and last (butlast x) = hd x by auto
obtain ps where ps;ps' =butlast x by auto
have ps∈C-star l
proof –
have length ps = l + 1 using ps ⟨length x = l + 2⟩ by auto
moreover have hd ps = hd x
  using ps ⟨length x = l + 2⟩
  by (metis (full-types) ⟨adjacent (last x) (hd x)⟩ adjacent-no-loop
app - Nil append-butlast-last-id butlast.simps(1) list.sel(1) hd-append2)
hence \(\text{adj-path} (\text{hd} \ ps) (\text{tl} \ ps)\) using \(\text{adj-path-buttlast}\)

by \((\text{metis} \quad \text{adj-path} (\text{hd} \ x) (\text{tl} \ x)\) (\text{butlast-tl} \ ps)\)

moreover have \(\text{last} \ ps = \text{hd} \ ps\)

by \((\text{metis} \quad \text{hd} \ ps = \text{hd} \ x) \quad \text{last} (\text{butlast} \ x) = \text{hd} \ x) \quad \text{ps}\)

ultimately show \(\text{thesis} using \ C\)-\text{star by auto}\)

qed

moreover have \(\text{ext} \ ps \ x\) using \(\text{ext}\)

by \((\text{metis} \quad \text{adj-path} (\text{hd} \ x) (\text{tl} \ x)\) (\text{adjacent} (\text{last} \ x) (\text{hd} \ x))\)

(last (butlast \ x) = \text{hd} \ x) adjacent-no-loop butlast \ simps(1) \ ps\)

ultimately show \(x \in (\bigcup ps \in (C\text{-star} \ l). \ \{ps', ext \ ps ps'\})\) by auto

qed

ultimately show \(\text{thesis by fast}\)

qed

ultimately show \(\text{thesis using card-partition}[\text{of} \ C\text{-star} \ l \ ext \ k] \ \langle k \geq 4 \rangle\)

by auto

qed

moreover have \(\text{card} \{ps. \ length \ ps = l+2 \land \text{adj-path} (\text{hd} \ ps) (\text{tl} \ ps) \land \text{adjacent} (\text{last} \ ps) (\text{hd} \ ps) \land \text{last} (\text{butlast} \ ps) \neq \text{hd} \ ps\} = \text{card} (T \ l - C\text{-star} \ l)\)

proof

obtain \(app where \ app \app=\langle \lambda ps. \ ps@SOME n. \text{adjacent} (\text{last} \ ps) n \land adjacent (\text{hd} \ ps) \ n\rangle\)

by auto

have \(\forall x. \ x \in app' (T \ l - C\text{-star} \ l) \implies x \in \{ps. \ length \ ps = l+2 \land \text{adj-path} (\text{hd} \ ps) (\text{tl} \ ps) \land \text{adjacent} (\text{last} \ ps) (\text{hd} \ ps) \land \text{last} (\text{butlast} \ ps) \neq \text{hd} \ ps\}\)

proof

fix \(x\) assume \(x \in app' (T \ l - C\text{-star} \ l)\)

then obtain \(ps where \ length \ ps = l+1 \ adj-path (\text{hd} \ ps) (\text{tl} \ ps) \land \text{last} \ ps \neq \text{hd} \ ps\)

\(x = \app \ ps\)

using \(T \ C\text{-star by auto}\)

hence \(last \ ps \in V\)

using \(\text{adj-path-V}[OF \quad \text{adj-path} (\text{hd} \ ps) (\text{tl} \ ps)]\)

by \((\text{cases} \ ps)\) auto

hence \(\exists n. \text{adjacent} (\text{last} \ ps) n \land \text{adjacent} (\text{hd} \ ps) \ n\)

using \(\text{adj-path-V}[OF \quad \text{adj-path} (\text{hd} \ ps) (\text{tl} \ ps)] \quad \text{last} \ ps \neq \text{hd} \ ps\)

[friend-assm[\text{of} \ last \ ps \ hd \ ps]]

by auto

moreover have \(\text{last} \ x = (\text{SOME} \ n. \text{adjacent} (\text{last} \ ps) n \land \text{adjacent} (\text{hd} \ ps) n)\)

using \(app (x = \app \ ps)\) by auto

ultimately have \(\text{adjacent} (\text{last} \ ps) (\text{last} \ x) \text{ and} \text{adjacent} (\text{hd} \ ps) (\text{last} \ x)\)

using \(\text{someI-ex by (metis (lifting))}\)+

have \(hd \ x = \text{hd} \ ps\) using \((x = \app \ ps) \quad \text{length} \ ps = l+1; \ app)\)

by \((\text{cases} \ ps)\) auto

have \(\text{length} \ x = l+2\) using \((x = \app \ ps) \quad \text{length} \ ps = l+1) \ app\ by auto

moreover have \(\text{adj-path} (\text{hd} \ x) (\text{tl} \ x)\)
proof 
  have last (tl ps)=last ps using ⟨length ps=l+1⟩
    by (metis last ps ≠ hd ps) list.sel(1,2) last-ConsL last-tl
neq-Nil-conv
moreover have length ps≠1 using ⟨last ps ≠ hd ps⟩
  by (metis Suc-eq-pluss1-left gen-length-code(1) gen-length-def
list.sel(1)
  last-ConsL length-Suc-conv neq-Nil-conv)
hence tl ps≠[] using ⟨length ps=l+1⟩
  by (metis add-diff-cancel-right' length-splice length-tl add.commute
  splice-Nil2)
ultimately have adj-path (hd ps) (tl ps &@[last x])
  using adj-path-app[OF ⟨adj-path (hd ps) (tl ps),of last x⟩
  ⟨adjacent (last ps) (last x)⟩:
    by auto
moreover have tl ps@[last x]=tl x
  using (x=app ps) app
    by (metis ⟨last x = (SOME n. adjacent (last ps) n ∧ adjacent
  (hd ps) n)⟩
    ⟨tl ps ≠ []⟩ list.sel(2) tl-append2)
ultimately show ?thesis using ⟨hd x=hd ps⟩ by auto
qed
moreover have adjacent (last x) (hd x)
  using ⟨hd x=hd ps⟩ ⟨adjacent (hd ps) (last x)⟩ adjacent-sym by auto
moreover have last (butlast x) ≠ hd x
  using ⟨last ps ≠ hd ps⟩ ⟨hd x=hd ps⟩
    by (metis ⟨x = app ps⟩ app butlast-snuc)
ultimately show length x = l + 2 ∧ adj-path (hd x) (tl x) ∧ adjacent
  (last x) (hd x)
    ∧ last (butlast x) ≠ hd x
  by auto
qed
moreover have ∃x. x∈{ps. length ps = l+2 ∧ adj-path (hd ps) (tl ps) ∧
  adjacent (last ps) (hd ps) ∧ last (butlast ps)≠hd ps}⇒ x∈app’(T l −
C-star l)
proof 
  fix x assume x∈{ps. length ps = l+2 ∧ adj-path (hd ps) (tl ps) ∧
  adjacent (last ps) (hd ps) ∧ last (butlast ps)≠hd ps}:
  hence length x=l+2 and adj-path (hd x) (tl x) and adjacent (last x)
(hd x)
  and last (butlast x)≠hd x
  by auto
  hence butlast x∈T l − C-star l
proof 
  have length (butlast x) = l + 1
    using ⟨length x = l + 2⟩ length-butlast by auto
moreover have hd (butlast x)=hd x
  using ⟨length x=l+2⟩
    by (metis append-butlast-last-id butlast.simps(1)) calculation
diff-add-inverse
diff-cancel2 hd-append length-butlast add.commute num.distinct(1)

one-eq-numeral-iff)
hence adj-path (hd (butlast x)) (tl (butlast x))
using adj-path (hd x) (tl x) by (metis adj-path-butlast butlast-tl)
moreover have last (butlast x) ≠ hd (butlast x)
using ⟨last (butlast x) ≠ hd x : hd (butlast x) = hd x⟩ by auto
ultimately show ?thesis using T C-star by auto
qed
moreover have app (butlast x) = x using app
proof –
have last (butlast x) ∈ V
proof (cases length x ≥ 3)
case True
hence last (butlast x) ∈ set (tl x)
proof (induct x)
case Nil
thus ?case by auto
next
case (Cons x1 t1)
have length t1 < 3 ⇒ ?case
proof –
assume length t1 < 3
hence length t1 = 2 using ⟨3 < length (x1 # t1)⟩ by auto
then obtain x2 t2 where t1 = x2 # t2 length t2 = 1
using Suc-length-conv[of 1 t1] by auto
then obtain x3 where t2 = [x3]
using Suc-length-conv[of 0 t2] by auto
have t1 = [x2, x3] using ⟨t1 = x2 # t2 ⋆ t2 = [x3]⟩ by auto
thus ?case by auto
qed
moreover have length t1 ≥ 3 ⇒ ?case
proof –
assume length t1 ≥ 3
hence last (butlast t1) ∈ set (tl t1)
using Cons.hyps by auto
thus ?case
by (metis butlast.simps(2) in-set-butlastD last.simps)
last-in-set
length-butlast length-greater-0-conv length-pos-if-in-set
length-tl list.sel(3))

qed
ultimately show ?case by force
qed
thus ?thesis using adj-path-V[OF (adj-path (hd x) (tl x))] by auto
next
case False
hence \(\text{length } x = 2\) using \(\langle \text{length } x = l + 2 \rangle\) by auto
then obtain \(x1\, x2\) where \(x = [x1, x2]\)
proof —
\begin{align*}
  \text{obtain } x1\, t1 \text{ where } &x = x1 \# t1 \text{ length } t1 = 1 \\
  \text{using } &\text{Suc-length-conv[of } 1\, x] \langle \text{length } x = 2 \rangle \text{ by auto} \\
  \text{then obtain } &x2 \text{ where } t1 = [x2] \\
  \text{using } &\text{Suc-length-conv[of } 0\, t1] \text{ by auto} \\
  \text{have } &x = [x1, x2] \text{ using } \langle x = x1 \# t1 \rangle \langle t1 = [x2] \rangle \text{ by auto} \\
  \text{thus } &?\text{thesis using that by auto} \\
  \text{qed}
\end{align*}

\begin{align*}
  \text{hence } &\text{last } (\text{butlast } x) = \text{hd } x \text{ by auto} \\
  \text{thus } &?\text{thesis using } \text{adj-path-V}[OF \langle \text{adj-path } (\text{hd } x) (tl\, x) \rangle] \text{ by auto} \\
  \text{auto} \\
  \text{moreover have } &\text{hd } (\text{butlast } x) = \text{hd } x \text{ using } \langle \text{length } x = l + 2 \rangle \text{ by } \langle \text{metis } \text{adjacent } (\text{last } x) (\text{hd } x) \rangle; \text{adjacent-no-loop append-butlast-last-id} \\
  \text{butlast.simps}(1) &\text{ list.sel}(1) \text{ hd-append} \\
  \text{hence } &\text{hd } (\text{butlast } x) \in V \text{ using } \text{adj-path-V}[OF \langle \text{adj-path } (\text{hd } x) (tl\, x) \rangle] \text{ by auto} \\
  \text{moreover have } &\text{last } (\text{butlast } x) \neq \text{hd } (\text{butlast } x) \\
  \text{using } &\langle \text{last } (\text{butlast } x) \rangle \neq \text{hd } (\text{butlast } x) = \text{hd } x \text{ by auto} \\
  \text{ultimately have } &\exists! n. \text{adjacent } (\text{last } (\text{butlast } x)) n \land \text{adjacent } (\text{hd } (\text{butlast } x)) n \\
  &\text{using friend-assm by auto} \\
  \text{moreover have } &\text{length } x \geq 2 \text{ using } \langle \text{length } x = l + 2 \rangle \text{ by auto} \\
  \text{hence } &\text{adjacent } (\text{last } (\text{butlast } x)) (\text{last } x) \\
  \text{using } &\langle \text{adj-path } (\text{hd } x) (tl\, x) \rangle; \text{by } \langle \text{induct } x, \text{auto}, \text{metis (full-types) adj-path.simps(2) append-Nil append-butlast-last-id, metis adj-path-app' append-butlast-last-id} \\
  \text{moreover have } &\text{adjacent } (\text{hd } (\text{butlast } x)) (\text{last } x) \\
  \text{using } &\langle \text{adjacent } (\text{last } x) (\text{hd } x) \rangle; \langle \text{hd } (\text{butlast } x) = \text{hd } x \rangle \text{ adjacent-sym by auto} \\
  \text{ultimately have } &\langle \text{SOME } n. \text{adjacent } (\text{last } (\text{butlast } x)) n \land \text{adjacent } (\text{hd } (\text{butlast } x)) n \rangle = \text{last } x \\
  \text{using } &\text{some1-equality by fast} \\
  \text{moreover have } &x = (\text{butlast } x)@[last\, x] \\
  \text{by } &\langle \text{metis adjacent } (\text{last } (\text{butlast } x)) (\text{last } x); \text{adjacent-no-loop append-butlast-last-id butlast.simps(1)} \rangle \\
  \text{ultimately show } &?\text{thesis using } \text{app by auto} \\
  \text{qed} \\
  \text{ultimately show } &x \in \text{app'}(T\, l - \text{C-star } l) \text{ by } \langle \text{metis image-iff} \rangle \\
  \text{qed} \\
  \text{ultimately have } &\text{app'}(T\, l - \text{C-star } l) = \{\text{ps. length } ps = l + 2 \land \text{adj-path} (\text{hd } ps) (tl\, ps) \land \text{adjacent } (\text{last } ps) (\text{hd } ps) \land \text{last } (\text{butlast } ps) \neq \text{hd } ps \} \text{ by fast} \\
  \text{moreover have } &\text{inj-on app } (T\, l - \text{C-star } l) \text{ using } \text{app unfolding inj-on-def} \text{ by auto} \\
  \text{ultimately show } &?\text{thesis by } \langle \text{metis card-image} \rangle
ultimately show \( \text{card} \ (C \ (l+1)) = k \times \text{card} \ (C \star l) + \text{card} \ (T \ l - C \star l) \) by auto

\[ \text{card} \ (C \ (l+1)) \equiv (k - (1::\text{nat})) \times \text{card} \ (C \star l) + \text{card} \ (T \ l - C \star l) \]

by auto
also have \( \ldots = k \times \text{card} \ (C \star l) + \text{card} \ (T \ l) - \text{card} \ (C \star l) \)
proof -
  have \( \text{card} \ (T \ l) \geq \text{card} \ (C \star l) \)
  using \( C \star l \subseteq T \ l \) (finite \( T \ l \)) by (metis card mono)
  thus \( \text{thesis} \) by auto

\[ \text{card} \ (C \ (l+1)) = k \times \text{card} \ (C \star l) + \text{card} \ (T \ l - C \star l) \]

by auto
also have \( \ldots = k \times \text{card} \ (C \star l) - \text{card} \ (C \star l) + \text{card} \ (T \ l) \)
proof -
  have \( \text{card} \ (T \ l) \geq \text{card} \ (C \star l) \)
  using \( C \star l \subseteq T \ l \) (finite \( T \ l \)) by (metis card mono)
  moreover have \( k \times \text{card} \ (C \star l) \geq \text{card} \ (C \star l) \) using \( k \geq 4 \) by auto
  ultimately show \( \text{thesis} \) by auto

\[ \text{card} \ (C \ (l+1)) = (k - (1::\text{nat})) \times \text{card} \ (C \star l) + \text{card} \ (T \ l) \]

by (metis monoid-mul-class.mult.left-neutral diff-mult-distrib)

finally have \( \text{card} \ (C \ (l+1)) = (k - (1::\text{nat})) \times \text{card} \ (C \star l) + \text{card} \ (T \ l) \)

\[ \text{card} \ (C \ (l+1)) \mod (k - (1::\text{nat})) = \text{card} (T \ l) \mod (k - (1::\text{nat})) \]

by (metis mod-mult-self3 mult.commute)
also have \( \ldots = (k \times (k-1) \mod (k - (1::\text{nat}))) \) using \( \text{T-count by auto} \)
also have \( \ldots = ((k \times (1::\text{nat}) + k) \mod (k - (1::\text{nat})) \)
proof -
  have \( k \times (k-1) \times (k+1) = (k - (1::\text{nat})) \times (k-1) \) using \( k \geq 4 \) by (metis diff-mult-distrib

\[ \text{card} \ (C \ (l+1)) \mod (k - (1::\text{nat})) = \text{card} (T \ l) \mod (k - (1::\text{nat})) \]

by (metis mod-mult-self3 mult.commute)
also have \( \ldots = (1 \mod (k - (1::\text{nat})) \)
by (metis mod-mult-right-eq mod-mult-mult-1 add.commute mult.commute)
also have \( \ldots = (k \mod (k - (1::\text{nat})) \) by auto
also have \( \ldots = (k \times (k-1) \mod (k - (1::\text{nat})) \) using \( k \geq 4 \) by auto
also have \( \ldots = (1 \mod (k - (1::\text{nat})) \) by (metis mod-mult-self2 add.commute

\[ \text{thesis} \) by auto

\[ \text{thesis} \) by auto

\[ \text{thesis} \) by auto

\[ \text{thesis} \) by auto

\[ \text{thesis} \) by auto

\[ \text{thesis} \) by auto

\[ \text{thesis} \) by auto

\[ \text{thesis} \) by auto

\[ \text{thesis} \) by auto
also have \( \ldots = l using \langle k \geq 4 \rangle \) by auto

finally show \( \text{card } (C (l+1)) \mod (k-(1::nat)) = 1 \).

qed

obtain \( p::nat \) where \( \text{prime } p \) \( p \text{ dvd } (k-(1::nat)) \) using \( \langle k \geq 4 \rangle \)
by (metis Suc-eq-plus1 Suc-numeral add-One-commute eq-iff le-diff-conv numeral-le-iff
one-le-numeral one-plus-BitM prime-factor-nat semiring-norm(69) semiring-norm(71))
hence \( p\text{-minus-1}:p-(1::nat)+1=p \)
by (metis add-diff-inverse add.commute not-less-iff-gr-or-eq prime-nat-def)
hence \( \ast: (\forall l::nat. \text{card } (C (l+1)) \mod p=1 \)
using \( (\forall l::nat. \text{card } (C (l+1)) \mod (k-(1::nat))=1) \mod-mod-cancel[OF \langle p \text{ dvd } (k-(1::nat))\rangle] \)
\langle prime p \rangle
by (metis mod-if prime-ge-1-nat)
have \( \text{card } (C (p - 1)) \mod p = 1 \)
proof (cases \( 2 \leq p \))
case True with \( \langle \text{of } p < 2 \rangle \) show \( ?\text{thesis} \)
by (metis \text{Nat.add-diff-assoc2 add-le-cancel-right diff-diff-left one-add-one p-minus-1)\)
next
case False with \( \langle \text{of } p < 2 \rangle \langle \text{prime } p \rangle \) \( \text{prime-ge-2-nat \ show \ ?\text{thesis} \) by blast
qed

moreover have \( \text{card } (C (p-(1::nat))) \mod p=0 \) using \( C \)
proof –
  have closure1:\( \forall x. x\in C (p-(1::nat))\implies \text{rotate1 } x \in C (p-(1::nat)) \)
  proof –
    fix \( x \)
    assume \( x\in C (p-(1::nat)) \)
    hence \( \text{length } x = p \) and \( \text{adj-path } (hd x) (tl x) \) and \( \text{adjacent } (\text{last } x) (hd x) \)
  using \( C \) \( p\text{-minus-1} \) by auto
  have \( \text{adjacent } (\text{last } (\text{rotate1 } x)) (hd (\text{rotate1 } x)) \)
  proof –
    have \( x\neq [] \) using \( \langle \text{length } x=p \rangle \langle \text{prime } p \rangle \) by auto
    hence \( \text{adjacent } (\text{last } (\text{rotate1 } x)) (hd (\text{rotate1 } x))=\text{adjacent } (hd x) (hd (tl x)) \)
  by (metis \langle \text{adjacent } (\text{last } x) (hd x) \) \langle \text{adjacent-no-loop append-Nil list.sel(1,3) hd-append2 last-snoc list.exhaust rotate1-hd-tl \rangle
also have \( \ldots = \text{True} \) using \( \langle \text{adj-path } (hd x) (tl x) \rangle \)
using \( \langle \text{adjacent } (\text{last } x) (hd x) \rangle \ associative \langle x \neq [] \rangle \)
by (metis \langle \text{adj-path,simps(2)} \) \langle \text{adjacent-no-loop append1-eq-conv append-Nil \rangle
append-Nil
append-butlast-last-id list.sel(1,3) list.exhaust)
finally show \( ?\text{thesis} \) by auto
qed

moreover have \( \text{adj-path } (hd (\text{rotate1 } x)) (tl (\text{rotate1 } x)) \)
proof –
  have \( x\neq [] \) using \( \langle \text{length } x=p \rangle \langle \text{prime } p \rangle \) by auto
then obtain \( y \, ys \) where \( y=\text{hd} \, x \, ys=\text{tl} \, x \) by auto

hence \( \text{adj-path} \, y \, ys \) and \( \text{adjacent} \) (last \( ys \)) \( y \) and \( ys\neq[] \)

by \( \text{metis} \, \langle \text{adj-path} \, (\text{hd} \, x) \rangle \, (\text{tl} \, x), \) \( \text{metis} \, \langle \text{adjacent} \, \text{(last} \, x) \rangle \, (\text{hd} \, x), \) \( \langle y \)

\( = \text{hd} \, x \)

(\( ys = \text{tl} \, x \)) \( \text{adjacent-no-loop} \, \text{list.sel}(1,3) \) \( \text{last.simps} \) \( \text{last-tl} \) \( \text{list.exhaust} \)

\( \text{list.sel}(1,3) \)

last-\text{ConsL} \, \text{neq}-\text{Nil-cone} \)

hence \( \text{adj-path} \, (\text{hd} \, (\text{rotate1} \, x)) \rangle \, (\text{tl} \, (\text{rotate1} \, x)) \)

\( =\text{adj-path} \, (\text{hd} \, (\text{ys}\circ[y])) \rangle \, (\text{tl} \, (\text{ys}\circ[y])) \)

using \( x\neq[] \, (y=\text{hd} \, x) \, (y=\text{tl} \, x) \) by \( \text{metis} \, \text{rotate1-1hd-tl} \)

also have \( ...=\text{adj-path} \, (\text{hd} \, (\text{ys})) \rangle \, (\text{tl} \, (\text{ys}\circ[y])) \)

by \( \text{metis} \, \langle y \neq [] \rangle \) \( \text{hd-append} \, \text{tl-append2} \)

also have \( ...=\text{True} \)

using \( \text{adj-path-app}[\text{OF} \, \text{adj-path} \, y \, ys; \, (ys\neq[]) \, \langle \text{adjacent} \, \text{(last} \, ys) \rangle \, y] \)

by \( \text{metis} \, \text{adj-path,simps}(2) \) \( \text{append-Cons} \, \text{list.sel}(1,3) \) \( \text{list.exhaust} \)

finally show \( \text{thesis} \) by auto

qed

moreover have \( \text{length} \, (\text{rotate1} \, x) \rangle = p \) using \( \langle \text{length} \, x=p \rangle \) by auto

ultimately show \( \text{rotate1} \, x \in C \, (p-(1::\text{nat})) \) using \( C \cdot p\text{-minus-1} \) by auto

qed

have closure:
\[
\bigwedge \, n \, x. \, x\in C \, (p-(1::\text{nat})) \implies \text{rotate} \, n \, x \in C \, (p-(1::\text{nat}))
\]

proof –

fix \( n \, x \) assume \( x\in C \, (p-(1::\text{nat})) \)

thus \( \text{rotate} \, n \, x \in C \, (p-(1::\text{nat})) \)

by \( \text{induct} \, n, \text{auto}, \text{metis} \, \text{One-nat-def closure1} \)

qed

obtain \( r \) where \( r\,r=\{\langle x,y \rangle. \, x\in C \, (p-(1::\text{nat})) \land (\exists \, n<p. \, \text{rotate} \, n \, x=y)\} \)

by auto

have \( \bigwedge \, x. \, x\in C \, (p-(1::\text{nat})) \implies p \text{ dvd} \, \{y, (\exists \, n<p. \, \text{rotate} \, n \, x=y)\} \)

proof –

fix \( x \) assume \( x\in C \, (p-(1::\text{nat})) \)

hence \( \text{length} \, x=p \) using \( C \cdot p\text{-minus-1} \) by auto

have \( \{y, (\exists \, n<p. \, \text{rotate} \, n \, x=y)\}=(\lambda n. \, \text{rotate} \, n \, x)^\prime \, \{0..<p\} \) by auto

moreover have \( \bigwedge \, n1 \, n2. \, n1\in\{0..<p\} \implies n2\in\{0..<p\} \implies n1\neq n2 \implies \text{rotate} \, n1 \, x\neq\text{rotate} \, n2 \, x \)

proof –

fix \( n1 \, n2 \)

assume \( n1\in\{0..<p\} \, n2\in\{0..<p\} \) \( \text{rotate} \, n1 \, x = \text{rotate} \, n2 \, x \, n1>n2 \)

obtain \( s::\text{nat} \) where \( s*(n1-n2) \text{ mod } p=1 \, s>0 \)

proof –

have \( n1-n2>0 \) and \( n1-n2<p \)

using \( n1\in\{0..<p\} \, n2\in\{0..<p\} \, n1>n2 \) by auto

hence \( \text{coprime} \, (n1-n2) \, p \) using \( \langle \text{prime} \, p \rangle \)

by \( \text{metis} \, \langle \text{full-types} \rangle \, \text{gcd-nat.commute} \, \text{nat-dvd-newless} \)

\text{prime-imp-coprime-nat} \)

105
hence \( \exists x, \left( (n1 \cdot n2) \cdot x = 1 \right) \mod p \) by (metis cong-solve-coprime-nat)
then obtain \( s :: \text{nat} \) where \( s \cdot (n1 \cdot n2) \mod p = 1 \)
by (metis \( \text{card} \) (\( C \) \( (p - (1::\text{nat})) \)) \mod p = 1; cong-nat-def
mod-mod-trivial
mult.commute)
moreover hence \( s > 0 \) by (metis mod-0 mult-0 neq0-conv
zero-neq-one)
ultimately show \( \text{thesis} \) using that by auto
qed
have rotate \( (s \cdot n1) \cdot x = \text{rotate} (s \cdot n2) \cdot x \)
using \( \text{rotate} \) \( n1 \cdot x = \text{rotate} \) \( n2 \cdot x \)
apply (induct \( s \))
apply (auto simp add: algebra-simps)
by (metis add.commute rotate-rotate
hence rotate \( (s \cdot n1 - s \cdot n2) \cdot x = x \)
using rotate-diff by auto
hence rotate \( (s \cdot (n1 - n2)) \cdot x = x \) by (metis diff-mult-distrib
mult.commute)
hence rotate \( 1 \cdot x = x \) using \( (s \cdot (n1 - n2)) \cdot \text{mod} p = 1 \); \( \text{length} \cdot x = p \)
by (metis rotate-conv-mod)
hence rotate \( 1 \cdot x = x \) by auto
have.hd \( x = \text{hd} \) \((\text{tl} \cdot x)\) using \( \text{prime} \) \( p \); \( \text{length} \cdot x = p \)
proof --
  have length \( x \geq 2 \) using \( \text{prime} \) \( p \); \( \text{length} \cdot x = p \) by auto
  hence length \( (\text{tl} \cdot x) \geq 1 \) by \text{force}
  hence \( x \neq [] \) and \( \text{tl} \cdot x \neq [] \) by auto+
  hence \( x = (\text{hd} \cdot x) # (\text{hd} \cdot (\text{tl} \cdot x)) # (\text{tl} \cdot (\text{tl} \cdot x)) \) using \( \text{hd} \cdot \text{Cons} \cdot \text{tl} \) by auto
hence \( (\text{hd} \cdot (\text{tl} \cdot x)) # (\text{tl} \cdot (\text{tl} \cdot x)) # (\text{tl} \cdot (\text{tl} \cdot x)) \) using \( \text{rotate} \cdot 1 \cdot x = x \) by (metis \( \text{Cons} \cdot \text{eq} \cdot \text{append} \cdot \text{tl} \) \( \text{rotate} \cdot 1 \cdot \text{simps} \) [2])
thus \( \text{thesis} \) by auto
qed
moreover have hd \( x \neq \text{hd} \) \((\text{tl} \cdot x)\)
proof --
have adj-path \( (\text{hd} \cdot x) \) \((\text{tl} \cdot x)\) using \( \text{x} \in C \) \( (p - (1::\text{nat})) \); \( C \) by auto
moreover have length \( x \geq 2 \) using \( \text{prime} \) \( p \); \( \text{length} \cdot x = p \) by auto
hence length \( (\text{tl} \cdot x) \geq 1 \) by \text{force}
  hence \( \text{tl} \cdot x \neq [] \) by \text{force}
ultimately have adjacent \( (\text{hd} \cdot x) \) \((\text{hd} \cdot (\text{tl} \cdot x))\)
by (metis \( \text{adj-path} \) \( \text{simps} \) [2] \text{list.sel} [1] \text{list.exhaust})
thus \( \text{thesis} \) by (metis adjacent-no-loop)
qed
ultimately have False by auto \}
thus False
by (metis \( \text{rotate} \) \( n1 \in \{0..p\} \); \( n1 \neq n2 \); \( n2 \in \{0..p\} \); \( \text{rotate} \) \( n1 \cdot x = \text{rotate} \) \( n2 \cdot x \)
less-linear)
qed
hence inj-on \( \lambda n. \text{rotate} \) \( n \cdot x \) \( \{0..p\} \) unfolding inj-on-def by fast
ultimately have \( \text{card} \) \( \{ y. \left( \exists n < p. \text{rotate} \) \( n \cdot x = y \) \} = \text{card} \) \( \{0..p\} \) by
(metis card-image)
  hence \( \text{card}\ \{y. (\exists n<p. \text{rotate}\ n\ x=y)\}=p \) by auto
thus \( p \text{ ded card}\ \{y. (\exists n<p. \text{rotate}\ n\ x=y)\} \) by auto
qed

hence \( \forall X \in C \ (p-(1::\text{nat})) \//\ r. \ p \text{ ded card} \ X \) unfolding quotient-def
Image-def \( r \) by auto
moreover have refl-\( (C\ (p-1))\) \( r \)
proof -
  have \( r \subseteq C\ (p-1) \times C\ (p-1) \)
proof
  fix \( x \) assume \( x \in r \)
  hence \( \text{fst}\ x \in C\ (p-1) \) and \( \exists n. \ \text{snd}\ x=\text{rotate}\ n\ (\text{fst}\ x) \) using \( r \) by auto
moreover then obtain \( n \) where \( \text{snd}\ x=\text{rotate}\ n\ (\text{fst}\ x) \) by auto
ultimately have \( \text{snd}\ x \in C\ (p-1) \) using closure by auto
moreover have \( x=(\text{fst}\ x,\text{snd}\ x) \) using \( x \in r \) \( r \) by auto
ultimately show \( x \in C\ (p-1) \times C\ (p-1) \) using \( \text{fst}\ x \in C\ (p-1) \)
by (metis SigmaI)
qed

moreover have \( \forall x \in C\ (p-1). \ (x, x) \in r \)
proof
  fix \( x \) assume \( x \in C\ (p-1) \)
  hence \( \text{rotate}\ \emptyset\ x \in C\ (p-1) \) using closure by auto
moreover have \( \emptyset<p \) using \( \prime\ p \) by auto
ultimately have \( (x,\text{rotate}\ \emptyset\ x) \in r \) using \( x \in C\ (p-1) \) \( r \) by auto
moreover have \( \text{rotate}\ \emptyset\ x=x \) by auto
ultimately show \( (x, x) \in r \) by auto
qed
ultimately show \( \text{thesis} \) using refl-on-def by auto
qed

moreover have sym \( r \) unfolding sym-def
proof (rule,rule,rule)
  fix \( x\ y \) assume \( (x, y) \in r \)
  hence \( x \in C\ (p-1) \) using \( r \) by auto
  hence length \( x=p \) using \( C\ p-\text{minus-1} \) by auto
obtain \( n \) where \( n<p \) rotate \( n\ x = y \) using \( (x,y) \in r \) \( r \) by auto
  hence \( y \in C\ (p-1) \) using closure[\( OF\ \{x \in C\ (p-1)\}\] by auto
have \( n=0 \Rightarrow (y, x) \in r \)
proof -
  assume \( n=0 \)
  hence \( x=y \) using \( \text{rotate}\ n\ x=y \) by auto
thus \( (y,x) \in r \) using refl-on-\( (C\ (p-1))\) \( r \) \( y \in C\ (p-1) \) refl-on-def
by fast
qed
moreover have \( n \neq 0 \Rightarrow (y,x) \in r \)
proof -
  assume \( n \neq 0 \)
  have \( \text{rotate}\ (p-n)\ y = x \)
proof
  have \( \text{rotate} \ (p - n) \ y = \text{rotate} \ (p - n) \ (\text{rotate} \ n \ x) \)
  using \( \langle \text{rotate} \ n \ x = y \rangle \) by auto
  also have \( \text{rotate} \ (p - n) \ (\text{rotate} \ n \ x) = \text{rotate} \ (p - n + n) \ x \)
  using rotate-rotate by auto
  also have \( \ldots = \text{rotate} \ p \ x \) using \( \langle n < p \rangle \) by auto
  also have \( \ldots = \text{rotate} \ 0 \ x \) using \( \langle \text{length} \ x = p \rangle \) by auto
  also have \( \ldots = x \) by auto
  finally show \(?thesis\) .
  qed
moreover have \( p - n < p \) using \( \langle n < p \rangle \) \( \langle n \neq 0 \rangle \) by auto
ultimately show \( (y, x) \in r \) by auto
qed
moreover have \( \text{trans} \ r \) unfolding trans-def
proof \( \langle \text{rule}, \text{rule}, \text{rule}, \text{rule}, \text{rule} \rangle \)
fix \( x \ y \ z \)
assume \( (x, y) \in r \) \( (y, z) \in r \)
hence \( x \in \text{C} \ (p - 1) \) using \( r \) by auto
hence \( \text{length} \ x = p \) using \( \text{C} \ p \)-minus-1 by auto
obtain \( n1 \ n2 \) where \( n1 < p \ n2 < p \ y = \text{rotate} \ n1 \ x \ z = \text{rotate} \ n2 \ y \)
using \( r \) \( \langle (x, y) \in r \rangle \) \( \langle (y, z) \in r \rangle \) by auto
hence \( z = \text{rotate} \ (n2 + n1) \ x \) by \( \langle \text{metis} \ \text{rotate-rotate} \rangle \)
hence \( z = \text{rotate} \ ((n2 + n1) \mod p) \ x \) using \( \langle \text{length} \ x = p \rangle \) by \( \langle \text{metis} \ \text{rotate-conv-mod} \rangle \)
moreover have \( (n2 + n1) \mod p < p \) by \( \langle \text{metis} \ \langle \text{prime} \ p \rangle \ \text{mod-less-divisor} \ \text{prime-gt-0-nat} \rangle \)
ultimately show \( (x, z) \in r \) using \( \langle x \in \text{C} \ (p - 1) \rangle \) \( r \) by auto
qed
moreover have \( \text{finite} \ \text{C} \ (p - 1) \)
by \( \langle \text{metis} \ \langle \text{card} \ (\text{C} \ (p - 1)) \ \text{mod} \ p = 1 \rangle \ \text{card-eq-0-iff} \ \text{mod-0} \ \text{zero-neq-one} \rangle \)
ultimately have \( p \ \text{dvd} \ \text{card} \ \langle \text{C} \ (p - (1::nat)) \rangle \) using \( \text{equiv-imp-dvd-card} \) \( \text{equiv-def} \) by fast
thus \( \text{card} \ (\text{C} \ (p - (1::nat))) \ \text{mod} \ p = 0 \) by \( \langle \text{metis} \ \text{dvd-eq-mod-eq-0} \rangle \)
qed
ultimately show \( \text{False} \) by auto
qed

theorem \( \langle \text{in valid-unSimpGraph} \rangle \) friendship-thm:
assumes \( \text{friend-assm}: \forall v \ u. \ v \in V \implies u \in V \implies v \neq u \implies \exists ! \ n. \ adjacent \ v \ n \wedge \adj u \ n \)
and \( \text{finite} \ V \)
shows \( \exists v. \ \forall n \in V. \ n \neq v \implies \text{adjacent} \ v \ n \)
proof
  have \( \text{card} \ V = 0 \implies \ ?thesis \)
  using \( \langle \text{finite} \ V \rangle \)
  by \( \langle \text{metis} \ \text{all-not-in-conv} \ \text{card-seteq} \ \text{empty-subsetI} \ \text{le0} \rangle \)
moreover have \( \text{card} \ V = 1 \implies \ ?thesis \)
proof
assume \( \text{card } V = 1 \)
then obtain \( v \) where \( V = \{ v \} \)
    using \( \text{card-eq-SucD[of } V \text{] by auto} \)
hence \( \forall n \in V. \ n = v \) by auto
thus \( \exists v. \forall n \in V. \ n \neq v \rightarrow \text{adjacent } v \ n \) by auto
qed
moreover have \( \text{card } V \geq 2 \implies \text{thesis} \)
proof –
    assume \( \text{card } V \geq 2 \)
hence \( \exists v \in V. \ \text{degree } v \ G = 2 \)
    using \( \text{exist-degree-two[of friend-assm] (finite } V \text{) by auto} \)
thus \( \text{thesis} \)
    using \( \text{degree-two-windmill[of friend-assm] (card } V \geq 2 \) (finite } V \text{) by auto} \)
qed
ultimately show \( \text{thesis by force} \)
qed
end

References


