The Königsberg Bridge Problem and the Friendship Theorem

Wenda Li
September 19, 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
11 Exclusive steps for combinatorial proofs

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 ←→ (u', w, v) ∈ edges G
  and no-id[simp]: (v, w, v) /∈ edges G

fun (in valid-unMultigraph) is-trail :: (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, 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{ v | v ∈ nodes g ∧ odd(degree v 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)

fun 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,e,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,e,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
  qed
by (metis (lifting, mono-tags) Collect-empty-eq assms(1) assms(2)
card-0-iff fst-conv mem-Collect-eq rev-finite-subset subsetI)
ultimately show ?thesis by arith
qed

finally have degree v (del-unEdge v e v' g) + 1 = card ({ea ∈ edges g. fst ea = v}) .
thus ?thesis unfolding degree-def .
qed

lemma del-edge-undirected-degree-plus: finite (edges g) ⇒ (v,e,v') ∈ edges g
⇒ (v',e,v) ∈ edges g ⇒ degree v' (del-unEdge v e v' g) + 1 = degree v' g
by (metis del-edge-undirected-degree-plus delete-edge-sym)

lemma del-edge-undirected-degree-minus[simp]: finite (edges g) ⇒ (v,e,v') ∈ edges g
⇒ (v',e,v) ∈ edges g ⇒ degree v (del-unEdge v e v' g) = degree v' g − (1::nat)
using del-edge-undirected-degree-plus by (metis add-diff-cancel-left add.commute)

lemma del-edge-undirected-degree-minus'[simp]: finite (edges g) ⇒ (v,e,v') ∈ edges g
⇒ (v',e,v) ∈ edges g ⇒ degree v' (del-unEdge v e v' g) = degree v g − (1::nat)
by (metis del-edge-undirected-degree-minus delete-edge-sym)

lemma del-unEdge-com: del-unEdge v w v' (del-unEdge n e n' g) = del-unEdge n e n' (del-unEdge v w v' g)
unfolding del-unEdge-def by auto

lemma rem-unPath-com: rem-unPath ps (del-unEdge v w v' g) = del-unEdge v w v' (rem-unPath ps g)
proof (induct ps arbitrary: g)
case Nil
thus ?case by (metis rem-unPath.simps(1))
next
case (Cons a ps')
thus ?case using del-unEdge-com
by (metis prod-cases3 rem-unPath.simps(1) rem-unPath.simps(2))
qed

lemma rem-unPath-valid[intro]:
valid-unMultigraph g ⇒ valid-unMultigraph (rem-unPath ps g)
proof (induct ps )
case Nil
thus ?case by simp
next
case (Cons x xs)
thus ?case
proof –
have \( \text{valid-unMultigraph} \) \((\text{rem-unPath} \ (x \neq \text{xs}) \ g) = \text{valid-unMultigraph} \)
\((\text{del-unEdge} \ (\text{fst} \ x) \ (\text{fst} \ (\text{snd} \ x)) \ (\text{snd} \ (\text{snd} \ x)) \ (\text{rem-unPath} \ \text{xs} \ g)) \)
using \( \text{rem-unPath-com} \) by \( \text{metis \ pair-collapse \ rem-unPath.simps(2)} \)
also have \( \ldots = \text{valid-unMultigraph} \) \((\text{rem-unPath} \ \text{xs} \ g) \)
by \( \text{metis \ Cons.hyps \ Cons.prems \ del-unEdge-valid} \)
also have \( \ldots = \text{True} \)
using \( \text{Cons by auto} \)
finally have \( \text{\?case = True} \).
thus \( \text{\?case by simp} \)
qed

\textbf{lemma (in valid-unMultigraph) degree-frame:}
\textbf{assumes} \( \text{finite \ (edges \ G) \ x \notin \{v, v'\}} \)
\textbf{shows} \( \text{degree} \ x \ (\text{del-unEdge} \ v \ w \ v' \ G) = \text{degree} \ x \ G \) \( \text{(is \ ?L=\?R)} \)
\textbf{proof (cases \( (v, w, v') \in \text{edges} \ G) \)}
\textbf{case True}
have \( ?L = \text{card} \{(e, e \in \text{edges} \ G - \{(v, w, v'),(v', w, v)\} \land \text{fst} \ e = x)\} \)
by \( \text{(simp add:del-unEdge-def degree-def)} \)
also have \( \ldots = \text{card} \{(e. e \in \text{edges} \ G \land \text{fst} \ e = x) - \{e. e \in \{(v, w, v'),(v', w, v)\} \land \text{fst} \ e = x\} \}
\)by \( \text{(metis \ set-compre-diff)} \)
also have \( \ldots = \text{card} \{(e. e \in \text{edges} \ G \land \text{fst} \ e = x)\} \) \( \text{using} \ x \notin \{v, v'\} \)
\textbf{proof –}
have \( x \neq v \land x \neq v' \) \( \text{using} \ x \notin \{v, v'\} \) \( \text{by \ simp} \)
hence \( \{e. e \in \{(v, w, v'),(v', w, v)\} \land \text{fst} \ e = x\} = \{\} \) \( \text{by \ auto} \)
thus \( \text{\?thesis by \ (metis \ Diff-empty)} \)
qed
also have \( \ldots = \?R \) \( \text{by \ (simp \ add:degree-def)} \)
finally show \( \text{\?thesis} \).
\textbf{next}
\textbf{case False}
moreover hence \( (v', w, v) \notin E \) \( \text{using} \ \text{corres by auto} \)
ultimately have \( E - \{(v, w, v'),(v', w, v)\} = E \) \( \text{by \ blast} \)
hence \( \text{del-unEdge} \ v \ w \ v' \ G = G \) \( \text{by \ (auto \ simp \ add:del-unEdge-def)} \)
thus \( \text{\?thesis by auto} \)
\textbf{qed}

\textbf{lemma [simp]: rev-path [] = [] unfolding rev-path-def by simp}
\textbf{lemma rev-path-append[simp]: rev-path (xs@ys) = (rev-path ys) @ (rev-path xs)}
\textbf{unfolding rev-path-def rev-append by auto}
\textbf{lemma rev-path-double[simp]: rev-path(rev-path xs)=xs}
\textbf{unfolding rev-path-def by (induct xs,auto)}

\textbf{lemma del-UnEdge-node[simp]: v\in\text{nodes} \ (del-unEdge \ u \ e \ v' \ G) \longleftrightarrow v\in\text{nodes} \ G}
\textbf{by \ (metis del-unEdge-def select-convs(1))}
lemma [intro!]: finite \((edges G)\) \implies \text{finite (} (\text{del-unEdge} u e u' G)\) 
   by (metis del-unEdge-def finite-Diff select-convs(2))

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

lemma [intro!]: finite \((edges G)\) \implies \text{finite (} (\text{del-unEdge} u e u' G)\) 
   by (metis del-unEdge-def finite-Diff select-convs(2))

proof (induct ps arbitrary:G)
  case Nil 
  thus ?case by simp

next 
  case (Cons x xs)
  hence finite \((edges (\text{rem-unPath} x#xs G))\) \implies \text{finite (} (\text{del-unEdge} v e v' G)\) 
     by (metis rem-unPath.simps(2) rem-unPath-com surjective-pairing)
  also have \(\ldots\) \text{finite (} (\text{rem-unPath} xs G)\) 
     by (metis finite emptyI finite-Diff2 finite-Diff-insert select-convs(2))
  also have \(\ldots\) \text{True using Cons by auto}
  finally have \(\ldots\) \text{True .}
  thus ?case by simp
qed

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

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

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

proof (induct ps)
  case Nil 
  show ?case by simp

next 
  case (Cons x xs)
  have nodes \((\text{rem-unPath} x#xs G)\)=nodes \((\text{del-unEdge} v e v' G)\) 
     by (metis rem-unPath.simps(2) rem-unPath-com surjective-pairing)
  also have \(\ldots\) \text{nodes \((\text{rem-unPath} xs G)\) by auto}
  also have \(\ldots\) \text{nodes \((\text{rem-unPath} xs G)\) using Cons by auto}
  finally show ?case .
qed

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

proof (induct ps)
  case Nil
  thus \(\text{case unfolding rev-path-def by auto}\)
next
  case (Cons x xs)
  obtain x1 x2 x3 where \(x:= (x1,x2,x3)\) by (metis prod-cases3)
  have set (rev-path \((x \neq xs)\)) = set ((rev-path xs)@\([x3,x2,x1]\))
    unfolding rev-path-def
    using x by auto
  also have \(\ldots= set (rev-path xs) \cup \{(x3,x2,x1)\}\) by auto
  finally have set (rev-path \((x \neq xs)\)) = set (rev-path xs) \cup \{(x3,x2,x1)\} .
  moreover have set \((x\neqxs)\) = set xs \cup \{(x1,x2,x3)\}
    by (metis List.set-simps(2) insert-is-Un sup-commute x)
  ultimately show \(\text{case using Cons by auto}\)
qed

lemma rem-unPath-edges:
  edges(rem-unPath ps G) = edges G - (set ps \cup set (rev-path ps))

proof (induct ps)
  case Nil
  show \(\text{case unfolding rev-path-def by auto}\)
next
  case (Cons x xs)
  obtain x1 x2 x3 where \(x:= (x1,x2,x3)\) by (metis prod-cases3)
  hence edges(rem-unPath \((x \neq xs)\) G) = edges(del-unEdge x1 x2 x3 (rem-unPath xs G))
    by (metis rem-unPath.simps(2) rem-unPath-com)
  also have \(\ldots= edges(rem-unPath xs G) - \{(x1,x2,x3),(x3,x2,x1)\}\)
    by (metis del-unEdge-def select-cons(2))
  also have \(\ldots= edges G - (set xs \cup set (rev-path xs)) - \{(x1,x2,x3),(x3,x2,x1)\}\)
    by (metis Cons.hyps)
  also have \(\ldots= edges G - (set (x\neqxs) \cup set (rev-path (x\neqxs)))\)
    proof -
    have set (rev-path xs) \cup \{(x3,x2,x1)\} = set ((rev-path xs)@\([x3,x2,x1]\))
      by (metis List.set-simps(2) empty-set set-append)
    also have \(\ldots= set (rev-path (x\neqxs))\) unfolding rev-path-def using x by auto
    finally have set (rev-path xs) \cup \{(x3,x2,x1)\} = set (rev-path (x\neqxs)) .
    moreover have set xs \cup \{(x1,x2,x3)\} = set \((x\neqxs)\)
      by (metis List.set-simps(2) insert-is-Un sup-commute x)
    moreover have edges G - (set xs \cup set (rev-path xs)) - \{(x1,x2,x3),(x3,x2,x1)\}\
      = edges G - ((set xs \cup \{(x1,x2,x3)\}) \cup (set (rev-path xs) \cup\\{(x3,x2,x1)\}))
      by auto
    ultimately show \(\text{thesis by auto}\)
  qed
next
  case Nil
  show \(\text{case .}\)
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 x)
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)) v = is-path v' ((rev-path xs) @ [(x3, x2, x1)])
v
  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 \(\ldots=\text{is-path } v (x\#xs) v'\)
  by (metis corres is-path.simps(1) is-path.simps(2) is-path-memb x)
finally have \(\text{is-path } v' (\text{rev-path } (x \# xs)) v=\text{is-path } v (x\#xs) v'\).
thus \(?case\).
qed

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

**lemma** \(\text{(in valid-unMultigraph) is-trail-path}:\)
\(\text{is-trail } v ps v' \iff \text{is-path } v ps v' \land \text{distinct } ps \land (\text{set } ps \cap \text{set } (\text{rev-path } ps) = \{\})\)
proof (induct ps arbitrary:v)
case Nil
show \(?case\) by auto
next
case \((\text{Cons } x xs)\)
obtain \(x_1 x_2 x_3\) where \(x: x=(x_1,x_2,x_3)\)
by (metis prod-cases3)
hence \(\text{is-trail } v (x\#xs) v' = (v=x_1 \land (x_1,x_2,x_3)\in E \land (x_1,x_2,x_3)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land \text{is-trail } x_3 xs v')\)
by (metis is-trail.simps(2))
also have \(\ldots=(v=x_1 \land (x_1,x_2,x_3)\in E \land (x_1,x_2,x_3)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land \text{is-path } x_3 xs v' \land \text{distinct } xs \land (\text{set } xs \cap \text{set } (\text{rev-path } xs) = \{\}))\)
using Cons.hyps by auto
also have \(\ldots=(\text{is-path } v (x\#xs) v' \land (x_1,x_2,x_3) \neq (x_3,x_2,x_1) \land (x_1,x_2,x_3)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land \text{distinct } (x\#xs)\land (\text{set } xs \cap \text{set } (\text{rev-path } xs) = \{\}))\)
by (metis append-Nil is-path.simps(1) is-path-split' no-id x)
also have \(\ldots=(\text{is-path } v (x\#xs) v' \land (x_1,x_2,x_3) \neq (x_3,x_2,x_1) \land (x_3,x_2,x_1)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land (x_3,x_2,x_1)\notin set xs \land \text{distinct } (x\#xs)\land (\text{set } xs \cap \text{set } (\text{rev-path } x\#xs) = \{\}))\)
proof –
  have \(\text{set } (\text{rev-path } (x\#xs)) = \text{set } ((\text{rev-path } xs)@[((x_3,x_2,x_1)])\) using x by auto
  also have \(\ldots=\text{set } (\text{rev-path } xs) \cup \{(x_3,x_2,x_1)\}\) by auto
  finally have \(\text{set } (\text{rev-path } (x\#xs))=\text{set } (\text{rev-path } xs) \cup \{(x_3,x_2,x_1)\}\).
thus \(?thesis\) by blast
qed
also have \(\ldots=(\text{is-path } v (x\#xs) v' \land \text{distinct } (x\#xs) \land (\text{set } (x\#xs) \cap \text{set } (\text{rev-path } (x\#xs)) = \{\}))\)
proof –
have \((x3,x2,x1) \notin \text{set } xs \iff (x1,x2,x3) \notin \text{set } (\text{rev-path } xs)\) using \text{in-set-rev-path}\n
by auto

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

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

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

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

thus \(?\text{thesis}\)

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

qed

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

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

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

qed

lemma (in \text{valid-unMultigraph}) \text{is-trail-rev}:

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

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

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

lemma (in \text{valid-unMultigraph}) \text{is-trail-intro}[intro]:

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

lemma (in \text{valid-unMultigraph}) \text{is-trail-split}:

\(\text{is-trail } v (p1 \oplus p2) v' \implies (\exists u. \text{is-trail } v p1 u \land \text{is-trail } u p2 v')\)

apply \text{induct } p1 \text{ arbitrary } v, \text{auto}

apply \text{metis } \text{is-trail-intro} \text{is-path-memb}

done

lemma (in \text{valid-unMultigraph}) \text{is-trail-split'}.\text{is-trail } v (p1 \oplus (u,w,u')\#p2) v' \implies \text{is-trail } v p1 u \land (u,w,u')\in E \land \text{is-trail } u' p2 v'

by (metis \text{is-trail}.\text{simps}(2) \text{is-trail-split})

lemma (in \text{valid-unMultigraph}) \text{distinct-elim}[simp]:

assumes \(\text{is-trail } v ((v1,w,v2)\#ps) v'\)

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

proof

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

thus \((v1,w,v2) \in E\) by (metis \text{assms} \text{is-trail}.\text{simps}(2))

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 \text{assms} \text{is-trail}.\text{simps}(2))

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

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

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

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

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

by (metis \text{rem-unPath}.\text{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'
shows valid-unMultigraph.is-trail G2 v ps v' using distinct-G1
proof (induct ps arbitrary: v)
case Nil
  hence v=v' ∧ v'∈nodes G1
  by (metis (full-types) assms (1) valid-unMultigraph.is-trail.simps (1))
  hence v=v' ∧ v'∈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∈edges G1
  by (metis Cons.prems assms (1) valid-unMultigraph.is-trail.simps (2) x)
moreover have v=x1 ∧ (x1;x2;x3)∉set xs ∧ (x3;x2;x1)∉set xs
  by (metis Cons.prems assms (1) valid-unMultigraph.is-trail.simps (2) x)
moreover have v=x1 (x1,x2,x3)∉set xs (x3,x2,x1)∉set xs by auto
ultimately show ?case by (metis assms (2) valid-unMultigraph.is-trail.simps (2) x)
qed

lemma (in valid-unMultigraph) distinct-path-intro':
assumes valid-unMultigraph.is-trail (rem-unPath p G) v ps v'
shows 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'
shows is-trail v ps v'
by (metis (full-types) assms distinct-path-intro' rem-unPath.simps (1)
    rem-unPath.simps (2))
lemma \textit{(in valid-unMultigraph) distinct-elim-rev}\texttt{[simp]}:
\begin{itemize}
  \item \texttt{assumes} is-trail \textit{v} ((\textit{v1}, \textit{w}, \textit{v2})\#\textit{ps}) \textit{v}'
  \item \texttt{shows} (\textit{v2}, \textit{w}, \textit{v1})\in\textit{edges}(\textit{rem-anPath ps G}) \iff (\textit{v2}, \textit{w}, \textit{v1})\in\textit{E}
\end{itemize}

\texttt{proof} –
\begin{itemize}
  \item \texttt{have} valid-unMultigraph \texttt{(rem-unPath ps G)} \texttt{using} valid-unMultigraph-axioms \texttt{by auto}
  \item \texttt{hence} (\textit{v2}, \textit{w}, \textit{v1})\in\textit{edges}(\textit{rem-unPath ps G}) \iff (\textit{v2}, \textit{w}, \textit{v1})\in\textit{E}
\end{itemize}

\texttt{by} \texttt{(metis valid-unMultigraph.corres)}

\texttt{moreover have} (\textit{v2}, \textit{w}, \textit{v1})\in\textit{E} \iff (\textit{v1}, \textit{w}, \textit{v2})\in\textit{E} \texttt{using} corres \texttt{by simp}

\texttt{ultimately show} \texttt{?thesis using} distinct-elim \texttt{by} \texttt{(metis assms)}

\texttt{qed}

\texttt{lemma (in valid-unMultigraph) del-UnEdge-even:}
\begin{itemize}
  \item \texttt{assumes} (\textit{v}, \textit{w}, \textit{v}') \in \textit{E} \texttt{finite} \textit{E}
  \item \texttt{shows} \textit{v} \in odd-nodes-set(del-unEdge \textit{v} \textit{w} \textit{v}' \textit{G}) \iff even (degree \textit{v} \textit{G})
\end{itemize}

\texttt{proof} –
\begin{itemize}
  \item \texttt{show} \texttt{?thesis} \texttt{by} \texttt{(metis (full-types) assms corres del-UnEdge-even delete-edge-sym)}
\end{itemize}

\texttt{qed}

\texttt{lemma (in valid-unMultigraph) del-UnEdge-even':}
\begin{itemize}
  \item \texttt{assumes} (\textit{v}, \textit{w}, \textit{v}') \in \textit{E} \texttt{finite} \textit{E}
  \item \texttt{shows} \textit{v}' \in odd-nodes-set(del-unEdge \textit{v} \textit{w} \textit{v}' \textit{G}) \iff even (degree \textit{v}' \textit{G})
\end{itemize}

\texttt{proof} –
\begin{itemize}
  \item \texttt{show} \texttt{?thesis} \texttt{by} \texttt{(metis (full-types) assms corres del-UnEdge-even delete-edge-sym)}
\end{itemize}

\texttt{qed}

\texttt{lemma del-UnEdge-even-even:}
\begin{itemize}
  \item \texttt{assumes} valid-unMultigraph \textit{G} \texttt{finite(edges G)} \texttt{finite(nodes G)} (\textit{v}, \textit{w}, \textit{v}')\in\textit{edges}
  \item \texttt{assumes} parity-assms: even (degree \textit{v} \textit{G}) even (degree \textit{v}' \textit{G})
  \item \texttt{shows} num-of-odd-nodes(del-anEdge \textit{v} \textit{w} \textit{v}' \textit{G})\equiv num-of-odd-nodes \textit{G} + 2
\end{itemize}

\texttt{proof} –
\begin{itemize}
  \item \texttt{interpret} \textit{G:valid-unMultigraph} \texttt{by fact}
  \item \texttt{have} \textit{v}\in\textit{odd-nodes-set(del-anEdge \textit{v} \textit{w} \textit{v}' \textit{G})}
  \item \texttt{by} \texttt{(metis \textit{G}.del-UnEdge-even assms(4) parity-assms(1))}
  \item \texttt{moreover have} \textit{v}'\in\textit{odd-nodes-set(del-anEdge \textit{v} \textit{w} \textit{v}' \textit{G})}
  \item \texttt{by} \texttt{(metis \textit{G}.del-UnEdge-even' assms(2) assms(4) parity-assms(2))}
  \item \texttt{ultimately have} extra-odd-nodes:\{\textit{v}, \textit{v}'\} \subseteq odd-nodes-set(del-anEdge \textit{v} \textit{w} \textit{v}' \textit{G})
  \item \texttt{unfolding} odd-nodes-set-def \texttt{by auto}
  \item \texttt{moreover have} \texttt{v} \notin odd-nodes-set \textit{G} \texttt{and} \texttt{v}'\notin odd-nodes-set \textit{G}
\end{itemize}

\texttt{qed}
using parity-assms unfolding odd-nodes-set-def by auto

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

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

proof
  fix \( x \)
  assume \( x\)-odd-set: \( x \in \text{odd-nodes-set } (\text{del-unEdge } v \ w \ v' \) \( G \)) \( - \{v, v'\} \)
  hence degree \( x \) (del-unEdge \( v \) \( w \) \( v' \) \( G \)) = degree \( x \) \( G \)
    by (metis Diff-iff \( G \).degree-frame 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 odd-nodes-set \( G \) \( \subseteq \) odd-nodes-set(del-unEdge \( v \) \( w \) \( v' \) \( G \))

proof
  fix \( x \)
  assume \( x\)-odd-set: \( x \in \text{odd-nodes-set } G \)
  hence \( x \not\in\{v, v'\} \implies \text{odd}(\text{degree } x \text{ (del-unEdge } v \ w \ v' \ G)) \)
    by (metis \( G \).degree-frame assms(2) mem-Collect-eq odd-nodes-set-def)
  hence \( x \not\in\{v, v'\} \implies x \in \text{odd-nodes-set } (\text{del-unEdge } v \ w \ v' \ G) \)
    using \( x\)-odd-set del-UnEdge-node unfolding odd-nodes-set-def by auto
  moreover have \( x \not\in\{v, v'\} \implies x \not\in \text{odd-nodes-set } (\text{del-unEdge } v \ w \ v' \ G) \)
    using extra-odd-nodes by auto
  ultimately show \( x \in \text{odd-nodes-set } (\text{del-unEdge } v \ w \ v' \ G) \) by auto
  qed

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

by auto

thus num-of-odd-nodes(del-unEdge \( v \) \( w \) \( v' \) \( G \)) = 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 \( v \not= v' \) using \( G \).no-id \( (v, w, v') \in \text{edges } G \) 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 \( G \).lifting assms(3) mem-Collect-eq odd-nodes-set-def rev-finite-subset subsetI)
  moreover have finite \( \{v, v'\} \) by auto
  ultimately show \(?thesis\) 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 \( G \)

13
interpret $G : \text{valid-unMultigraph}$ by fact
have $\text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v') \ G$
  by (metis $G.\text{del-UnEdge-even}$ assms(2) assms(4) parity-assms(1))
have $\text{not-odd-v'} : v' \notin \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v') \ G$
  by (metis $G.\text{del-UnEdge-even}$ assms(2) assms(4) parity-assms(2))
have $\text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v') \cup \{v\} \subseteq \text{odd-nodes-set} \cup \{v\}$
proof
  fix $x$
  assume $x$-prems: $x \in \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v') \cup \{v\}$
  have $x \Rightarrow x \in \text{odd-nodes-set} \cup \{v\}$
    using parity-assms
  by (metis (lifting) $G.E.\text{validD}(2)$ Un-def assms(4) mem-Collect-eq odd-nodes-set-def
       Un-def assms(2) sup-commute)
moreover have $x \Rightarrow x \in \text{odd-nodes-set} \cup \{v\}$
  by (metis insertI1 insert-is-Un simp)
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 $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} \text{ unfolding odd-nodes-set-def}$
    using $G.\text{degree-frame} : \text{finite (edges G)}$ by auto
  hence $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} \cup \{v\}$ by simp
  ultimately show $x \in \text{odd-nodes-set} \cup \{v\}$ by auto
qed
moreover have $\text{odd-nodes-set} \cup \{v\} \subseteq \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v' \ G)$
proof
  fix $x$
  assume $x$-prems: $x \in \text{odd-nodes-set} \cup \{v\}$
  have $x \Rightarrow x \in \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v' \ G) \cup \{v\}$
    by (metis UnI1 odd-v)
moreover have $x \Rightarrow x \in \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v' \ G) \cup \{v\}$
  by auto
moreover have $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} \cup \{v\}$ using $x$-prems by auto
  hence $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v' \ G) \text{ unfolding odd-nodes-set-def}$
    using $G.\text{degree-frame} : \text{finite (edges G)}$ by auto
  hence $x \notin \{v, v'\} \Rightarrow x \in \text{odd-nodes-set} \cup \{v\}$ by simp
  ultimately show $x \in \text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v' \ G) \cup \{v\}$ by auto
qed
ultimately have $\text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v' \ G) \cup \{v\} = \text{odd-nodes-set} \cup \{v\}$
by auto
moreover have $\text{odd-nodes-set} \cap \{v\} = \{\}$
  using parity-assms unfolding odd-nodes-set-def by auto
moreover have $\text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v' \ G) \cap \{v'\} = \{\}$
  by (metis Int-insert-left-if0 inf-bot-left inf-commute not-odd-v')
moreover have $\text{finite} (\text{odd-nodes-set}(\text{del-unEdge} \ v \ w \ v' \ G))$
  using finite (nodes G) by auto
moreover have finite (odd-nodes-set G) using (finite (nodes G)) by auto
ultimately have \( \text{card}(\text{odd-nodes-set G}) + \text{card} \{v\} = \text{card}(\text{odd-nodes-set(del-unEdge v w v') G}) + \text{card} \{v'\} \)
using card-Un-disjoint[of odd-nodes-set (del-unEdge v w v' 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 G = num-of-odd-nodes(del-unEdge v w v' 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-subsetI 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 \((\text{del-unEdge } v w v') G) \subseteq \text{odd-nodes-set } G\)

proof
fix \(x\)
assume \(x\)-odd-set: \(x \in \text{odd-nodes-set } (\text{del-unEdge } v w v') G\)
hence \(x \notin \{v,v'\} \Rightarrow \text{odd}(\text{degree } x G)\)
using assms \(G\).degree-frame unfolding odd-nodes-set-def
by auto
hence \(x \notin \{v,v'\} \Rightarrow x \in \text{odd-nodes-set } G\)
using \(x\)-odd-set del-UnEdge-node unfolding odd-nodes-set-def
by auto
moreover have \(x \in \{v,v'\} \Rightarrow x \in \text{odd-nodes-set } G\)
using extra-odd-nodes by auto
ultimately show \(x \in \text{odd-nodes-set } G\)
by auto
qed

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

proof
assume odd-nodes-set \(G = \text{odd-nodes-set } (\text{del-unEdge } v w v') G \cup \{v,v'\}\)
misove have odd-nodes-set \((\text{del-unEdge } v w v') G \cap \{v,v'\} = \{\}\)
using vv'-disjoint by auto
moreover have finite(odd-nodes-set \((\text{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 \(\text{card}(\text{odd-nodes-set } G)\)
unfolding num-of-odd-nodes-def
using \(\text{card-Un-disjoint}\)
by metis
moreover have \(v \neq v'\) using \(G\).no-id \((v,w,v') \in \text{edges } G\)
by auto
hence \(\text{card}\{v,v'\}=2\) by simp
ultimately show \(\text{?thesis}\) unfolding num-of-odd-nodes-def by simp
qed

\[\text{lemma } (\text{in valid-unMultigraph} ) \text{ rem-UnPath-parity-v'}:\]
assumes finite \(E\) is-trail \(v ps v'\)
shows \(v \neq v' \iff (\text{odd } (\text{degree } v' (\text{rem-unPath } ps G)) = \text{even} (\text{degree } v' G))\)
using assms
proof (induct ps arbitrary: v)
case Nil
thus \(\text{?case}\) by (metis is-trail.simps(1) rem-unPath.simps(1))
next
case (Cons \(x\) \(xs\)) print-cases
obtain \(x1 x2 x3\) where \(x = (x1,x2,x3)\) by (metis prod-cases3)
hence rem-x:odd \((\text{degree } v'(\text{rem-unPath } (x\#xs) G)) = \text{odd}(\text{degree } v'(\text{del-unEdge } x1 x2 x3 (\text{rem-unPath } xs G)))\)
by \((\text{metis } \text{rem-unPath}.\text{simps}(2) \text{ rem-unPath-com})\)

have \(x3 = v' \implies ?\text{case}\)

proof \((\text{cases } v=v')\)

case True

assume \(x3 = v'\)

have \(x1 = v'\) using \(x\) by \((\text{metis Cons.prems(2) True is-trail}.\text{simps}(2))\)

thus \(?\text{thesis}\) using \(x3 = v'\) by \((\text{metis Cons.prems(2) is-trail}.\text{simps}(2) \text{ no-id } x)\)

next

case False

assume \(x3 = v'\)

have \(\text{odd } (\text{degree } v' (\text{rem-unPath } (x \neq xs) G)) = \text{odd } (\text{degree } v' (\text{del-unEdge } x1 x2 x3 \text{ (rem-unPath } x G))))\) using rem-x.

also have \(\ldots = \text{odd } (\text{degree } v' (\text{rem-unPath } x G) - 1)\)

proof

have \(\text{finite } (\text{edges } (\text{rem-unPath } x G))\)

by \((\text{metis } \text{full-types } \text{assms}(1) \text{ finite-Diff rem-unPath-edges})\)

moreover have \((x1,x2,x3) \in \text{edges}( \text{rem-unPath } x G)\)

by \((\text{metis Cons.prems(2) distinct-elim is-trail}.\text{simps}(2) x)\)

moreover have \((x3,x2,x1) \in \text{edges}( \text{rem-unPath } x G)\)

by \((\text{metis Cons.prems(2) corres distinct-elim-rev is-trail}.\text{simps}(2) x)\)

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

by \((\text{metis } x3 = v' \text{ del-edge-undirected-degree-minus delete-edge-sym } x)\)

qed

also have \(\ldots = \text{even } (\text{degree } v' (\text{rem-unPath } x G))\)

proof

have \((x1,x2,x3) \in E\) by \((\text{metis Cons.prems(2) is-trail}.\text{simps}(2) x)\)

hence \((x3,x2,x1) \in \text{edges}( \text{rem-unPath } x G)\)

by \((\text{metis Cons.prems(2) corres distinct-elim-rev x})\)

hence \((x3,x2,x1) \in \{ e \in \text{edges } (\text{rem-unPath } x G). \text{fst } e = v' \}\)

using \(x3 = v'\) by \((\text{metis } \text{mono-tags} \text{ fst-conv mem-Collect-eq})\)

moreover have \(\text{finite } \{ e \in \text{edges } (\text{rem-unPath } x G). \text{fst } e = v' \}\)

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

ultimately have \(\text{degree } v' (\text{rem-unPath } x G) \neq 0\)

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

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

qed

also have \(\ldots = \text{even } (\text{degree } v' G)\)

using \(x3 = v'\) \text{assms}

by \((\text{metis } \text{mono-tags} \text{ Cons.hyps Cons.prems(2) is-trail}.\text{simps}(2) x)\)

finally have \(\text{odd } (\text{degree } v' (\text{rem-unPath } (x \neq xs) G)) = \text{even } (\text{degree } v' G)\)

thus \(?\text{thesis}\) by \((\text{metis False})\)

qed

moreover have \(x3 \neq v' \implies ?\text{case}\)

proof \((\text{cases } v=v')\)

case True

assume \(x3 \neq v'\)

have \(\text{odd } (\text{degree } v' (\text{rem-unPath } (x \neq xs) G)) = \text{odd } (\text{degree } v' (\text{del-unEdge } x1 x2 x3 \text{ (rem-unPath } x G))))\) using rem-x.
also have \(\ldots = \text{odd}(\text{degree } v') (\text{rem-unPath } xs \; G) - 1\)

proof

- have 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 True x
  by (metis Cons.prems(2) del-edge-undirected-degree-minus is-trail.simps(2))

qed

also have \(\ldots = \text{even}(\text{degree } v') (\text{rem-unPath } xs \; G)\)

proof

- have \((x_1, x_2, x_3) \in E\) by (metis Cons.prems(2) is-trail.simps(2) x)

hence \((x_1, x_2, x_3) \in \text{edges} (\text{rem-unPath } xs \; G)\)
  by (metis Cons.prems(2) distinct-elim x)

hence \((x_1, x_2, x_3) \in \{ e \in \text{edges} (\text{rem-unPath } xs \; G). \text{fst } e = v' \}\)
  using \(v = v' \; \text{Cons}\)

  by (metis (lifting, mono-tags) fst-conv is-trail.simps(2) mem-Collect-eq)

moreover have finite \\{ e \in \text{edges} (\text{rem-unPath } xs \; G). \text{fst } e = v' \\}
  using 'finite E' by auto

ultimately have degree v' (rem-unPath xs G) \(\neq 0\)
  unfolding degree-def by auto

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

qed

also have \(\ldots \neq \text{even } (\text{degree } v' G)\)

using \(x_3 \neq v'\) assms

by (metis Cons.hyps Cons.prems(2) is-trail.simps(2) x)

finally have odd (degree v’ (rem-unPath (x \# xs) G)) \(\neq\) even (degree v’ G).

thus \(?\text{thesis}\) by (metis True)

next

case \(\text{False}\)

assume \(x_3 \neq v'\)

have odd \(\text{degree } v' (\text{rem-unPath } (x \# xs) \; G)\) = odd (degree v’ (del-unEdge x_1 x_2 x_3 (rem-unPath \; G)))
  using rem-x.

also have \(\ldots = \text{odd}(\text{degree } v' (\text{rem-unPath } xs \; G))\)

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 \(x_3 \neq v'\) empty-iff insert-iff)

moreover have valid-unMultigraph \((\text{rem-unPath } xs \; G)\)
  using valid-unMultigraph-axioms by auto

moreover have finite \((\text{edges } (\text{rem-unPath } xs \; G))\)
  by (metis (full-types) assms(1) finite-Diff rem-unPath-edges)

ultimately have degree v’ (del-unEdge x_1 x_2 x_3 (rem-unPath \; G))
  = degree v' (rem-unPath xs \; G)
  using degree-frame

by (metis valid-unMultigraph.degree-frame)
thus ?thesis by simp

qed

also have \ldots = even (degree v' G)
  using assms \langle x3 \neq v' \rangle
  by (metis Cons.hyps Cons.prems(2) is-trail.simps(2))

finally have odd (degree v' (rem-unPath (x # xs) G)) = even (degree v' G).
  thus ?thesis by (metis False)

qed

ultimately show ?case by auto

qed

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

lemma (in valid-unMultigraph) rem-UnPath-parity-others:
  assumes finite E is-trail v ps v'
  shows even (degree n (rem-unPath ps G)) = even (degree n G)
  using assms proof
    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)
    hence even (degree n (rem-unPath (x#xs) G)) = even (degree n (del-unEdge x1 x2 x3 (rem-unPath xs G)))
      by (metis rem-unPath.simps(2) rem-unPath-com)
    have n=x3 \implies ?case
      proof
        assume n=x3
        have even (degree n (rem-unPath (x#xs) G)) = even (degree n (del-unEdge x1 x2 x3 (rem-unPath xs G)))
          by (metis rem-unPath.simps(2) rem-unPath-com x)
        also have \ldots = even (degree n (rem-unPath xs G) - 1)
          proof
            have finite (edges (rem-unPath xs G))
              by (metis (full-types) assms(1) finite-Diff rem-unPath-edges)
            moreover have \langle x1,x2,x3 \rangle \subseteq edges (rem-unPath xs G)
              by (metis Cons.prems(2) distinct-elim is-trail.simps(2) x)
            moreover have \langle x3,x2,x1 \rangle \subseteq edges (rem-unPath xs G)
              by (metis Cons.prems(2) corres distinct-elim-rev is-trail.simps(2) x)
            ultimately show \ldots
              using (n = x3) del-edge-undirected-degree-minus'
                by auto
          qed
          qed
        also have \ldots = odd (degree n (rem-unPath xs G))
          proof
            have (x1,x2,x3) \in E by (metis Cons.prems(2) is-trail.simps(2) x)
hence \((x_3, x_2, x_1) \in \text{edges}(\text{rem-unPath} \; \text{xs} \; G)\) 
by \((\text{metis} \; \text{Cons} \; \text{prems}(2) \; \text{corres} \; \text{distinctelimrev} \; x)\) 

hence \((x_3, x_2, x_1) \in \{e \in \text{edges}(\text{rem-unPath} \; \text{xs} \; G). \; \text{fst} \; e = n\}\) 
using \((n=x_3) \; \text{by} \; (\text{metis} \; (\text{mono-tags}) \; \text{fstconv} \; \text{memCollecteq})\) 

moreover have finite \(\{e \in \text{edges}(\text{rem-unPath} \; \text{xs} \; G). \; \text{fst} \; e = n\}\) 
using \((\text{finite} \; E) \; \text{by} \; \text{auto})\) 
ultimately have degree \(n \; (\text{rem-unPath} \; \text{xs} \; G) \neq 0\) 

unfolding degree-def by \(\text{auto}\) 
thus ?thesis by \(\text{auto}\) 
qed 
also have ...=even(degree \(n \; G)\) 
proof – 
have \(x_3 \neq v^{'}) by \((\text{metis} \; (n = x_3) \; \text{assms}(3) \; \text{insertiff})\) 
hence odd \(\text{degree} \; x_3 \; (\text{rem-unPath} \; \text{xs} \; G)) = \text{even}(\text{degree} \; x_3 \; G)\) 
using Cons assms 
by \((\text{metis} \; \text{is-trail}.\text{.simps}(2) \; \text{rem-UnPath-parity-v} \; x)\) 
thus ?thesis using \((n=x_3) \; \text{by} \; \text{auto})\) 
qed 
finally have even \(\text{degree} \; n \; (\text{rem-unPath} \; (x \# \text{xs}) \; G) = \text{even}(\text{degree} \; n \; G)\) . 
thus ?thesis . 
qed 
moreover have \(n \neq x_3 \implies ?case\) 
proof – 
assume \(n \neq x_3\) 
have even \(\text{degree} \; n \; (\text{rem-unPath} \; (x \# \text{xs}) \; G) = \text{even}(\text{degree} \; n \; (\text{del-unEdge} \; x_1 \; x_2 \; x_3 \; (\text{rem-unPath} \; \text{xs} \; G))))\) 
by \((\text{metis} \; \text{rem-unPath}.\text{.simps}(2) \; \text{rem-unPath-com} \; x)\) 
also have ...=even(degree \(n \; (\text{rem-unPath} \; \text{xs} \; G)\)) 
proof – 
have \(v = x_1\) by \((\text{metis} \; \text{cons}.\text{prems}(2) \; \text{is-trail}.\text{.simps}(2) \; x)\) 
hence \(n \notin \{x_1, x_3\}\) by \((\text{metis} \; \text{cons}.\text{prems}(3) \; (n \neq x_3) \; \text{insertE} \; \text{insertII} \; \text{singletonE})\) 

moreover have valid\text{-}\text{Multigraph} \; (\text{rem-unPath} \; \text{xs} \; G) 
using valid\text{-}\text{Multigraph-axioms} \; \text{by} \; \text{auto} 

moreover have finite \((\text{edges} \; (\text{rem-unPath} \; \text{xs} \; G))\) 
by \((\text{metis} \; \text{full-types} \; \text{assms}(1) \; \text{finiteDiffs} \; \text{rem-unPath-edges})\) 
ultimately have degree \(n \; (\text{del-unEdge} \; x_1 \; x_2 \; x_3 \; (\text{rem-unPath} \; \text{xs} \; G)) = \text{degree} \; n \; (\text{rem-unPath} \; \text{xs} \; G)\) using degree-frame 
by \((\text{metis} \; \text{valid-\text{Multigraph}.degree-frame})\) 
thus ?thesis by \(\text{simp}\) 
qed 
also have ...=even(degree \(n \; G)\) 
using Cons assms \((n \neq x_3) \; x\) by \(\text{auto}\) 
finally have even \(\text{degree} \; n \; (\text{rem-unPath} \; (x \# \text{xs}) \; G)) = \text{even}(\text{degree} \; n \; G)\) . 
thus ?thesis . 
qed 
ultimately show ?case by \(\text{auto}\) 
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) ∧ 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
proof
  case (Cons x xs)
  have x-in: (x1, x2, x3) ∈ edges (rem-unPath xs G)
    by (metis full-types Cons.prems assms)
  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))
  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) ∧ x3 ≠ v' then 2 else 0) + 2
    using Cons.prems[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) ∧ x3 ≠ v' \iff 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
      have x1 ≠ x3 by (metis valid-rem-xs valid-unMultigraph.no-id x-in)
      moreover hence x1 ≠ v'
      using Cons assms
      by (metis is-trail.simps(2) parity-x1-x3(1) rem-UnPath-parity-v' x)
    qed
qed
ultimately have \( x_1 \notin \{x_3, v' \} \) by auto

hence even\((\text{degree } x_1 G)\)

using \(\text{Cons.prems}(3)\) \(\text{assms}(1)\) \(\text{assms}(2)\) \(\text{parity-x1-x3}(1)\)

by \((\text{metis \,(full-types)}\) \(\text{is-trail.simps}(2)\) \(\text{rem-UnPath-parity-others } x)\)

hence even\((\text{degree } x_1 G) \land x_1 \neq v'\) using \(x_1 \neq v'\) by auto

hence even\((\text{degree } v G) \land v \neq v'\) by \((\text{metis \,Cons.prems}(3)\) \(\text{is-trail.simps}(2)\)

\(x)\)

thus \(?thesis\) by auto

qed

finally have num-of-odd-nodes\((\text{rem-unPath} (x \# xs) G)\)

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

thus \(?thesis\).

qed

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

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

proof –

assume parity-x1-x3: even\((\text{degree } x_1 (\text{rem-unPath} xs G))\)

odd\((\text{degree } x_3 (\text{rem-unPath} xs G))\)

have num-of-odd-nodes\((\text{rem-unPath} (x \# xs) G)\)= num-of-odd-nodes\((\text{del-UnEdge} x_1 x_2 x_3 (\text{rem-unPath} xs G))\)

by \((\text{metis \,rem-unPath.simps}(2)\) \(\text{rem-unPath-com } x)\)

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

using \(\text{parity-x1-x3} \) \(\text{fin-nodes} \) \(\text{fin-edges} \) \(\text{valid-rem-xs} \) \(\text{x-in} \)

by \((\text{metis \,del-UnEdge-even-odd})\)

also have \(\ldots =\text{num-of-odd-nodes} G + (\text{if even} (\text{degree } x_3 G) \land x_3 \neq v' \text{ then } 2 \text{ else } 0)\)

using \(\text{Cons.hyps \,Cons.prems}(3)\) \(\text{assms}(1)\) \(\text{assms}(2)\) \(\text{parity-assms } x\)

by auto

also have \(\ldots =\text{num-of-odd-nodes} G + 2\)

proof –

have even\((\text{degree } x_3 G) \land x_3 \neq v' \text{ \(\iff\)} odd\((\text{degree } x_3 (\text{rem-unPath} xs G))\)

using \(\text{Cons.prems} \) \(\text{assms}\)

by \((\text{metis \,is-trail.simps}(2)\) \(\text{parity-x1-x3}(2)\) \(\text{rem-UnPath-parity-v } x)\)

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

qed

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

proof –

have \(x_1 \neq x_3\) by \((\text{metis \,valid-rem-xs} \) \(\text{valid-unMultigraph.} \text{no-id } x\text{-in})\)

moreover hence \(x_1 \neq v'\)

using \(\text{Cons} \) \(\text{assms}\)

by \((\text{metis \,is-trail.simps}(2)\) \(\text{parity-x1-x3}(1)\) \(\text{rem-UnPath-parity-v' } x)\)

ultimately have \(x_1 \notin \{x_3, v' \}\) by auto

hence even\((\text{degree } x_1 G)\)

using \(\text{Cons.prems}(3)\) \(\text{assms}(1)\) \(\text{assms}(2)\) \(\text{parity-x1-x3}(1)\)

by \((\text{metis \,(full-types)}\) \(\text{is-trail.simps}(2)\) \(\text{rem-UnPath-parity-others } x)\)

hence even\((\text{degree } x_1 G) \land x_1 \neq v'\) using \(x_1 \neq v'\) by auto

hence even\((\text{degree } v G) \land v \neq v'\) by \((\text{metis \,Cons.prems}(3)\) \(\text{is-trail.simps}(2)\)

22
thus \( ?\text{thesis} \) by auto
qed

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

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

moreover have odd (degree x1 (rem-unPath xs G)) \implies
even (degree x3 (rem-unPath xs G)) \implies \text{?case}
proof
assume parity-x1-x3: odd (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)
using parity-x1-x3 fin-nodes fin-edges valid-rem-xs x-in
by (metis del-UnEdge-odd-even)
also have ...
num-of-odd-nodes G + (if even (degree x3 G) \land x3 \neq v' then 2 else 0)

proof (cases v \neq v')
case True
have x1 \neq 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))
\iff odd(degree x1 G)
using True parity-x1-x3(1) rem-UnPath-parity-others x Cons.prems(3)
assms(1) assms(2) by auto

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

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

24
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)

next
case Nil
thus ?case by auto

next
case (Cons x xs)
obtain x1 x2 x3 where x = (x1, x2, x3)
by (metis prod-cases)

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))
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 odd (degree x3 G) ∧ x3 ≠ v' then −2 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

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

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 \( v \neq v' \))
  case True
  have \( x1 \neq x3 \) by (metis valid-rem-xs valid-unMultigraph.no-id x-in)
  moreover have \( \text{is-trail} \ x3 \ x \ x' \)
    by (metis Cons.prems(3) is-trail.simps(2) x)
  ultimately have \( \text{even} \ (\text{degree} \ x1 \ (\text{rem-unPath} \ x G)) \leftrightharpoons \text{even} \ (\text{degree} \ x1 \ G) \)
    using True Cons.prems(3) assms(2) parity-x1-x3(1)
    rem-UnPath-parity-others x
  by auto
  hence \( \text{even} \ (\text{degree} \ x1 \ G) \) by (metis parity-x1-x3(1))
  thus \( \text{thesis} \) by (metis hide-lams, mono-tags Cons.prems(3) is-trail.simps(2)
    monoid-add-class.add.right-neutral x)

next
  case False
  then show \( \text{thesis} \) by auto
qed
finally have \( \text{num-of-odd-nodes} \ (\text{rem-unPath} \ (x\#xs) \ G) = \text{num-of-odd-nodes} \ G + (\text{if odd}(\text{degree} \ v \ G) \land v \neq v' \text{ then } -2 \text{ else 0}) \)
thus \( \text{thesis} \).

qed

moreover have \( \text{even} \ (\text{degree} \ x1 \ (\text{rem-unPath} \ x \ G)) \implies \text{odd} \ (\text{degree} \ x3 \ (\text{rem-unPath} \ x \ G)) \implies \text{thesis} \)
proof
  assume parity-x1-x3: \( \text{even} \ (\text{degree} \ x1 \ (\text{rem-unPath} \ x \ G)) \)
    \( \text{odd} \ (\text{degree} \ x3 \ (\text{rem-unPath} \ x \ G)) \)
  have \( \text{num-of-odd-nodes} \ (\text{rem-unPath} \ (x\#xs) \ G) = \text{num-of-odd-nodes} \)
    \( (\text{del-UnEdge} \ x1 \ x2 \ x3 \ (\text{rem-unPath} \ x \ G)) \)
    by (metis rem-unPath.simps(2) rem-unPath-com x)
  also have \( \ldots = \text{num-of-odd-nodes} \ (\text{rem-unPath} \ x \ G) \)
    using parity-x1-x3 simp(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 Cons.hyps[\( \langle \text{finite} \ E \rangle \langle \text{finite} \ V \rangle \), of x3] Cons.prems(3) assms(1)
    assms(2)
    parity-assms x
  by auto
  also have \( \ldots = \text{num-of-odd-nodes} \ G \)
    proof
      have \( \text{odd}(\text{degree} \ x3 \ G) \land x3 \neq v' \implies \text{even} \ (\text{degree} \ x3 \ (\text{rem-unPath} \ x \ G)) \)
      using Cons.prems assms
      by (metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x)
thus \( \text{thesis using parity-x1-x3(2) by auto} \)

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 \( v \neq v' \))

case True

\( \text{have x1} \neq \text{x3 by (metis valid-rem-xs valid-unMultigraph.no-id x-in)} \)

moreover have \( \text{is-trail x3 xs v'} \) \( \text{by (metis Cons.prems(3) is-trail.simps(2) x)} \)

ultimately have \( \text{even (degree x1 (rem-unPath xs } G)) \)

\( \Longleftrightarrow \) \( \text{even (degree x1 } G) \)

using True Cons.prems(3) assms(1) assms(2) parity-x1-x3(1)

rem-UnPath-parity-others \( x \) \( \text{by auto} \)

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

with Cons.prems(3) \( x \) show \( \text{thesis by auto} \)

next

case False

then show \( \text{thesis by auto} \)

qed

finally have \( \text{num-of-odd-nodes (rem-unPath x\#xs } G) = \)

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

thus \( \text{thesis} \).

qed

moreover have \( \text{odd (degree x1 (rem-unPath xs } G)) \)

\( \Longrightarrow \) \( \text{even (degree x3 (rem-unPath xs } G)) \)

\( \Longrightarrow \) \( \text{thesis} \)

proof

assume parity-x1-x3: \( \text{odd (degree x1 (rem-unPath xs } G)) \)

\( \text{even (degree x3 (rem-unPath xs } G)) \)

have \( \text{num-of-odd-nodes (rem-unPath x\#xs } G) = \text{num-of-odd-nodes} \)

\( \text{(del-UnEdge x1 x2 x3 (rem-unPath xs } G)) \)

\( \text{by (metis rem-unPath.simps(2) rem-unPath-com x)} \)

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

using parity-x1-x3 fin-nodes fin-edges valid-rem-xs x-in

\( \text{by (metis del-UnEdge-odd-even)} \)

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 Cons.hyps Cons.prems(3) assms(1) assms(2) parity-assms x

by auto

also have \( \ldots = \text{num-of-odd-nodes } G + ( -2) \)

proof

have \( \text{odd}(\text{degree x3 } G) \land x3 \neq v' \) \( \Longleftrightarrow \) \( \text{even (degree x3 (rem-unPath xs } G)) \)

using Cons.prems assms

by (metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x)

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

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

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

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 \textit{is-trail}.simp\$s(2) parity-x1-x3(1) rem-UnPath-parity-v' x)
ultimately have \( x_1 \notin \{x_3, v'\} \) by auto
hence odd\((\text{degree } x_1 G)\)
using Cons assms\$s(3) assms(1) assms(2) parity-x1-x3(1)
by \textit{is-trail}.simp\$s(2) rem-UnPath-parity-others x
hence odd\((\text{degree } x_1 G) \land x_1 \neq v' \) using \( x_1 \neq v' \) by auto
hence odd\((\text{degree } v G) \land v \neq v' \) by (metis Cons assms\$s(3) \textit{is-trail}.simp\$s(2)

thus \(?thesis\) by auto
qed

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

proof

assume parity-x1-x3: odd\((\text{degree } x_1 (\text{rem-unPath} \ x G))
odd\((\text{degree } x_3 (\text{rem-unPath} \ x G))
have num-of-odd-nodes\((\text{rem-unPath} \ 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)
also have ...=\text{num-of-odd-nodes} \ (\text{rem-unPath} \ x G)-(2::nat)

using \textit{del-UnEdge-odd-odd}
by (metis \textit{add-implies-diff} \textit{fin-edges} \textit{fin-nodes} parity-x1-x3 valid-rem-xs x-in)

also have ...=\text{num-of-odd-nodes} \ G-(2::nat)

proof

have odd\((\text{degree } x_3 G) \land x_3 \neq v' \leftrightarrow \text{even}(\text{degree } x_3 (\text{rem-unPath} \ x G))
using Cons assms\$s(3) assms(1) assms(2) parity-x1-x3(2)
by (metis \textit{is-trail}.simp\$s(2) rem-UnPath-parity-v x)
hence \neg(\text{odd}(\text{degree } x_3 G) \land x_3 \neq v') by (metis parity-x1-x3(2)
have num-of-odd-nodes\((\text{rem-unPath} \ x G))= \text{num-of-odd-nodes} \ G+(\text{if odd}(\text{degree } x_3 G) \land x_3 \neq v' \text{ then } -2 \text{ else } 0)
by (metis Cons assms\$s(3) assms(1) assms(2) \textit{is-trail}.simp\$s(2) parity-assms x)
thus \(?thesis\)

using \neg(\text{odd}(\text{degree } x_3 G) \land x_3 \neq v') by auto
qed

also have ...=\text{num-of-odd-nodes} \ G+(\text{if 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 \( x \neq v' \)
using Cons assms

by (metis is-trail.simps(2) parity-x1-x3(1) rem-UnPath-parity-v x)

ultimately have \( x \notin \{x_3, v'\} \) by auto

hence \( \text{odd}(\text{degree} x G) \)
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 \( \text{odd}(\text{degree} x G) \land x \neq v' \) using \( x \neq v' \)

by auto

hence \( \text{odd}(\text{degree} v G) \land v \neq v' \) by (metis Cons.prems(3) is-trail.simps(2) x)

hence \( v \in \text{odd-nodes-set} G \)
using Cons.prems(3) E-validD(1) 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 (degree v G) \& v \neq v')

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

moreover have \( \text{finite}(\text{odd-nodes-set} G) \)
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 (odd (degree v G) \& v \neq v') by auto

qed

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

thus ?thesis .

qed

ultimately show ?case by metis

qed

lemma (in valid-unMultigraph) rem-UnPath-cycle:
assumes \( \text{finite} E \text{ finite} V \text{ is-trail} v \text{ ps} v' = v' \)
shows \( \text{num-of-odd-nodes} (\text{rem-unPath} \text{ ps} G) = \text{num-of-odd-nodes} G \) (is \( L=R \))

proof (cases even(\( \text{degree} v' G)\))

case True

hence \( \text{num-of-odd-nodes} G + (\text{if even}(\text{degree} v G) \land v \neq v' \text{ then } 2 \text{ else } 0) \)

by (metis assms(1) assms(2) assms(3) rem-UnPath-even)

with assms show ?thesis by auto

next
case False

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

by (metis assms(1) assms(2) assms(3) rem-UnPath-odd)

thus ?thesis using (v = v') by auto

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 –
  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

ultimately show ?case using Cons by auto

thus ∃ps. is-trail v ps v' by rule

qed

lemma (in valid-unMultigraph) no-rep-length: is-trail v ps v'⇒length ps=card(set ps)

by (induct ps arbitrary:v, auto)

lemma (in valid-unMultigraph) path-in-edges: is-trail v ps v'⇒set ps ⊆ 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 ⊆ E using Cons by auto

moreover have x∈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 ≤ 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 ⇒ nat ⇒ bool where

exist-path-length v l≡∃v' ps. is-trail v' ps v ∧ length ps=l

lemma (in valid-unMultigraph) longest-path:

assumes finite E n ∈ V

shows ∃v. ∃max-path. is-trail v max-path n ∧

(∀v'. ∀e∈E. ¬is-trail v' (e#max-path) n)

proof (rule ccontr)

assume contro:¬ (∃v max-path. is-trail v max-path n ∧

(∀v'. ∀e∈E. ¬is-trail v' (e#max-path) n))

hence induct:(∀v max-path. is-trail v max-path n

⇒ (∃v'. ∃e∈E. is-trail v' (e#max-path) n)) by auto

have is-trail n [] n using ⟨n ∈ V⟩ by auto

hence exist-path-length n 0 unfolding exist-path-length-def by auto

moreover have ∀y. exist-path-length n y → y ≤ card E

using trail-bound[OF ⟨finite E⟩] unfolding exist-path-length-def by auto

hence bound:∀y. exist-path-length n y → y < 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
  hence length max-path + 1 \( \leq (\text{GREATEST} \; x. \; \text{exist-path-length} \; n \; x) \) 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 \( \langle \text{even} (\text{card} \; A) \rangle \) 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 \( (\text{degree} \; v' \; G) \; \land \; v' \neq v) \; \lor \; (odd \; (\text{degree} \; v' \; G) \; \land \; v' = v) \)
  shows \( \exists e \; v1. \; \text{is-trail} \; v1 \; (e \# \text{ps} \; v) \)
proof
  have \( (\text{even} \; (\text{degree} \; v' \; G) \; \land \; v' \neq v) \; \Longrightarrow \; \text{odd} \; (\text{degree} \; v' \; (\text{rem-unPath} \; ps \; G)) \)
    by (metis assms (1) assms (2) rem-UnPath-parity-v)
  moreover have \( (\text{odd} \; (\text{degree} \; v' \; G) \; \land \; v' = v) \; \Longrightarrow \; \text{odd} \; (\text{degree} \; v' \; (\text{rem-unPath} \; ps \; G)) \)
    by (metis assms (1) assms (2) rem-UnPath-parity-v)
  ultimately have \( \text{odd} \; (\text{degree} \; v' \; (\text{rem-unPath} \; ps \; G)) \) using parity-assms by auto
  hence odd (card \( \{ e. \; \text{fst} \; e = v' \; \land \; e \in \text{edges} \; G - (\text{set} \; ps \; \cup \; \text{set} \; (\text{rev-path} \; ps)) \} \))
    by (metis (lifting, no-types) Collect-cong)
  hence \( \{ e. \; \text{fst} \; e = v' \; \land \; e \in E - (\text{set} \; ps \; \cup \; \text{set} \; (\text{rev-path} \; ps)) \} = \{ \})
    by (metis empty_iff finite.emptyI odd-card)
  then obtain \( v0 \) \( w \) where \( v0w: \; (v', w, v0) \in E \; (v', w, v0) \notin \text{set} \; ps \; \cup \; \text{set} \; (\text{rev-path} \; ps) \) by auto
hence \( \text{is-trail} \ v \emptyset \ ((v_0,w,v')\#\text{ps}) \ v \)
by \( \text{metis} \ (\text{hide-lams}, \text{mono-tags}) \ \text{Un-iff} \ \text{assms}(2) \ \text{corres} \ \text{in-set-rev-path} \ \text{is-trail}.\text{simps}(2) \)

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

\text{qed}

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

\begin{verbatim}
fun replace-by-UnPath :: '(v,'w) path ⇒ 'v × 'w × 'v ⇒ ('v,'w) path ⇒ ('v,'w)
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)
\end{verbatim}

\text{lemma} (in \text{valid-unMultigraph}) \text{del-unEdge-connectivity}:
\text{assumes} connected \( ∃ \text{ps}. \ \text{valid-graph.is-path} \ (\text{del-unEdge} \ v \ e \ v' \ G) \ v \ \text{ps} \ v' \)
\text{shows} \text{valid-unMultigraph.connected} \ (\text{del-unEdge} \ v \ e \ v' \ G)

\text{proof} –
\begin{itemize}
  \item \text{have} \text{valid-unMulti}::valid-unMultigraph \ (\text{del-unEdge} \ v \ e \ v' \ G)
    \text{using} \text{valid-unMultigraph-axioms by simp}
  \item \text{have} \text{valid-graph}::valid-graph \ (\text{del-unEdge} \ v \ e \ v' \ G)
    \text{using} \text{valid-graph-axioms del-undirected by} \ (\text{metis delete-edge-valid})
  \item \text{obtain} \text{ex-path} where \text{ex-path}::valid-graph.is-path \ (\text{del-unEdge} \ v \ e \ v' \ G) \ v \ \text{ex-path} \ v'
    \text{by} \ (\text{metis} \ \text{assms}(2))
  \item \text{show} \ ?\text{thesis} unfolding \text{valid-unMultigraph.connected-def} [OF \text{valid-unMulti}]
\end{itemize}
\text{proof} (\text{rule,rule,rule})
\begin{itemize}
  \item \text{fix} \ n \ n'
  \item \text{assume} \ n : \ n \in nodes \ (\text{del-unEdge} \ v \ e \ v' \ G)
  \item \text{assume} \ n' : \ n' \in nodes \ (\text{del-unEdge} \ v \ e \ v' \ G)
  \item \text{assume} \ n \neq n'
  \item \text{obtain} \text{ps} where \text{ps}::\text{is-path} \ n \ \text{ps} \ n'
    \text{by} \ (\text{metis} \ (\text{n}\neq\text{n'}). \ n' \in nodes; \ \text{connected-def}; \ \text{del-UnEdge-node})
  \item \text{hence} \text{valid-graph.is-path} \ (\text{del-unEdge} \ v \ e \ v' \ G)
    \text{by} \ (\text{metis} \ \text{assms}(1) \ n' \ \text{replace-by-UnPath}.\text{simps}(1))
  \item \text{valid-graph}
    \text{valid-graph.is-path-simps}(1)
\end{itemize}
\text{next}
\begin{itemize}
  \item \text{case} (\text{Cons} \ x \ \text{xs})
  \item \text{obtain} \ x1 \ x2 \ x3 \ where \ x::x=(x1,x2,x3)
    \text{by} \ (\text{metis} \ \text{prod-cases3})
  \item \text{have} \ x=(v,e,v') \ \Longrightarrow \ ?\text{case}
  \item \text{proof} –
    \item \text{assume} \ x=(v,e,v')
    \item \text{hence} \text{valid-graph.is-path} \ (\text{del-unEdge} \ v \ e \ v' \ G)
\end{itemize}
\[ n \text{ (replace-by-UnPath } (x \#xs) (v, e, v') \text{ ex-path) } n' \]  
\[ = \text{valid-graph.is-path (del-Edge v e v' G)} \]  
\[ n \text{ (ex-path@(replace-by-UnPath xs (v, e, v') ex-path)) } n' \]  
\[ \text{by (metis replace-by-UnPath.simps(2))} \]  
\[ \text{also have } \ldots = \text{True} \]  
\[ \text{by (metis Cons.hyps Cons.prems (x = (v, e, v)): ex-path is-path.simps(2))} \]  
\[ \text{valid-graph} \]  
\[ \text{valid-graph.is-path-split} \]  
\[ \text{finally show } \text{thesis by simp} \]  
\[ \text{qed} \]  
\[ \text{moreover have } x = (v', e, v) \Rightarrow \text{?case} \]  
\[ \text{proof} - \]  
\[ \text{assume } x = (v', e, v) \]  
\[ \text{hence valid-graph.is-path (del-Edge v e v' G)} \]  
\[ n \text{ (replace-by-UnPath } (x \#xs) (v, e, v') \text{ ex-path) } n' \]  
\[ = \text{valid-graph.is-path (del-Edge v e v' G)} \]  
\[ n \text{ ((rev-path ex-path)@(replace-by-UnPath xs (v, e, v') ex-path)) } n' \]  
\[ \text{by (metis Cons.prems is-path.simps(2) no-id replace-by-UnPath.simps(2))} \]  
\[ \text{also have } \ldots = \text{True} \]  
\[ \text{by (metis Cons.hyps Cons.prems (x = (v', e, v)): is-path.simps(2))} \]  
\[ \text{ex-path valid-graph} \]  
\[ \text{valid-graph.is-path-split valid-unMulti valid-unMultigraph.is-path-rev} \]  
\[ \text{finally show } \text{thesis by simp} \]  
\[ \text{qed} \]  
\[ \text{moreover have } x \neq (v', e, v) \land x \neq (v', e, v) \Rightarrow \text{?case} \]  
\[ \text{by (metis Cons.hyps Cons.prems del-Edge-frame is-path.simps(2))} \]  
\[ \text{replace-by-UnPath.simps(2)} \]  
\[ \text{valid-graph valid-graph.is-path.simps(2) x) \]  
\[ \text{ultimately show } \text{?case by auto} \]  
\[ \text{qed} \]  
\[ \text{thus } \exists \text{ ps. valid-graph.is-path (del-Edge v e v' G) n ps n' by auto} \]  
\[ \text{qed} \]  
\[ \text{qed} \]

**Lemma (in valid-unMultigraph) path-between-odds:**

**Assumes**
- odd\((\text{degree } v G)\)
- odd\((\text{degree } v' G)\)
- finite\(E\)
- \(v \neq v'\)
- num-of-odd-nodes\(G = 2\)

**Shows**
- \(\exists \text{ ps. is-trail } v \text{ ps } v'\)

**Proof**

- **have** \(v \in V\)
  - **proof** (rule ccontr)
    - **assume** \(v \notin V\)
    - **hence** \(\forall e \in E. fst e \neq v\) by (metis E-valid(1) imageI set-mp)
    - **hence** \(\text{degree } v G = 0\) unfolding degree-def using \(\text{finite } E\)
    - **by force**
    - **thus** False using \((\text{odd}(\text{degree } v G))\) by auto
  - **qed**
- **have** \(v' \in V\)
  - **proof** (rule ccontr)
    - **assume** \(v' \notin V\)
hence $\forall e \in E. \; \text{fst } e \neq v'$ by (metis E-valid(1) image1 set-mp)

hence degree $v' \in G = 0$ unfolding degree-def using (finite E)

by force

thus False using (odd (degree $v' \in G$)) by auto

qed

then obtain max-path $v0$ where max-path:

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

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

have even (degree $v0 \in G$) $\implies v0 = v' \implies v0 = v$

by (metis assms(2))

moreover have even (degree $v0 \in G$) $\implies v0 = v' \implies v0 = v$

proof

- assume even (degree $v0 \in G$) $v0 \neq v'$

hence $\exists w \in V. \; \text{is-trail } v1 (w \# \text{max-path } v')$

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

thus $\text{thesis}$ by (metis (full-types) is-trail.simps(2) max-path(2) prod.exhaust)

qed

moreover have odd (degree $v0 \in G$) $\implies v0 = v' \implies v0 = v$

proof

- assume odd (degree $v0 \in G$) $v0 = v'$

hence $\exists w \in V. \; \text{is-trail } v1 (w \# \text{max-path } v')$

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

thus $\text{thesis}$ by (metis (full-types) List.set-simps(2) insert-subset max-path(2) path-in-edges)

qed

moreover have odd (degree $v0 \in G$) $\implies v0 \neq v' \implies v0 = v$

proof (rule ccontr)

- assume $v0 \neq v$ odd (degree $v0 \in G$) $v0 \neq v'$

moreover have $v \in \text{odd-nodes-set } G$

using ($v \in V$. odd (degree $v \in G$)) unfolding odd-nodes-set-def

by auto

moreover have $v \in \text{odd-nodes-set } G$

using ($v' \in V$. odd (degree $v' \in G$)) unfolding odd-nodes-set-def

by auto

ultimately have $\{v, v', v0\} \subseteq \text{odd-nodes-set } G$

using is-path-memb[OF is-trail-intro[OF is-trail v0 max-path v']] max-path(1)

unfolding odd-nodes-set-def

by auto

moreover have $\text{card } \{v, v', v0\} = 3$ using ($v0 \neq v$) ($v \neq v'$) ($v0 \neq v'$) by auto

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


by auto

ultimately have $3 \leq \text{card } (\text{odd-nodes-set } G)$ by (metis card_mono)

thus False using (num-of-odd-nodes $G = 2$) unfolding num-of-odd-nodes-def

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

lemma (in valid-unMultigraph) del-unEdge-even-connectivity:
assumes finite \( E \) finite \( V \) connected \( \forall n \in V.\ even(\text{degree } n \ G) \ (v,e,v') \in E\)
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)
have fin-\( E' \): finite(edges (del-unEdge \( v \ e v' \ G \))
  by (metis (hide-tams, no-types) assms(1) del-undirected delete-edge-def
     finite-Diff select-convs(2))
have fin-\( V' \): finite(nodes (del-unEdge \( v \ e v' \ G \))
  by (metis (mono-tags) assms(2) del-undirected delete-edge-def select-convs(1))
have all-even: \( \forall n \in \text{nodes} (\text{del-unEdge} \ v \ e v' \ G), \ n \notin \{v,v'\} \)
  \( \rightarrow \) even(\text{degree } n \ (\text{del-unEdge} \ v \ e v' \ G))
  by (metis (full-types) assms(4) degree-frame del-UnEdge-node)
moreover have even (\text{degree } v \ G) by (metis (full-types) E-validD(1) assms(4) assms(5))
moreover have even (\text{degree } v' \ G) by (metis (full-types) E-validD(2) assms(4)
     assms(5))
moreover have num-of-odd-nodes \( G = 0 \)
  using \( \forall n \in V.\ even(\text{degree } n \ G) \)\ (finite \( V \))
unfolding num-of-odd-nodes-def odd-nodes-set-def by auto
ultimately have num-of-odd-nodes (del-unEdge \( v \ e v' \ G \)) = 2
  using del-UnEdge-even-even[of \( G \ v \ e v' \ OF \) valid-unMultigraph-axioms]
  by (metis assms(1) assms(2) assms(5) monoid-add-class.add.left-neutral)
moreover have odd (\text{degree } v \ (\text{del-unEdge} \ v \ e v' \ G))
  using (even (\text{degree } v \ G) \)\ del-UnEdge-even\( \langle v,v',v' \rangle \in E \)\ (finite \( E \))
unfolding odd-nodes-set-def
  by auto
moreover have odd (\text{degree } v' \ (\text{del-unEdge} \ v \ e v' \ G))
  using (even (\text{degree } v' \ G) \)\ del-UnEdge-even\( \langle v,v',v' \rangle \in E \)\ (finite \( E \))
unfolding odd-nodes-set-def
  by auto
moreover have finite (\text{edges} (\text{del-unEdge} \ v \ e v' \ G))
  using (finite \( E \)) by auto
moreover have \( v \neq v' \) using no-id \( \langle v,v',v' \rangle \in E \) by auto
ultimately have \( \exists \text{ps. valid-unMultigraph.is-trail} (\text{del-unEdge} \ v \ e v' \ G) \) \( v \) \( \text{ps} \) \( v' \)
  using valid-unMultigraph.path-between-odds\( \langle \text{OF valid-unMulti,of } v v' \rangle \)
  by auto
thus \(?thesis\)
  by (metis (full-types) assms(3) del-unEdge-connectivity valid-unMulti
     valid-unMultigraph.is-trail-intro)
qed
lemma (in valid-graph) path-end: ps ≠ [] → is-path v ps v' → v' = snd (snd (last ps))
  by (induct ps arbitrary: v, auto)

lemma (in valid-unMultigraph) connectivity-split:
  assumes connected ~valid-unMultigraph.connected (del-unEdge v w v' G)
  (v, w, v') ∈ E
  obtains G1 G2 where
    nodes G1 = {n. ∃ ps. valid-graph.is-path (del-unEdge v w v' G) n ps v}
    and edges G1 = {(n, e, n'). (n, e, n') ∈ edges (del-unEdge v w v' G)
                     ∧ n ∈ nodes G1 ∧ n' ∈ nodes G1}
    and nodes G2 = {n. ∃ ps. valid-graph.is-path (del-unEdge v w v' G) n ps v'}
    and edges G2 = {(n, e, n'). (n, e, n') ∈ edges (del-unEdge v w v' G)
                     ∧ n ∈ nodes G2 ∧ n' ∈ nodes G2}
    and edges G1 ∪ edges G2 = edges (del-unEdge v w v' G)
    and edges G1 ∩ edges G2 = {}
    and nodes G1 ∪ nodes G2 = nodes (del-unEdge v w v' G)
    and nodes G1 ∩ 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. ∃ ps. valid-graph.is-path (del-unEdge v w v' G) n ps v}
    by metis
  then obtain G1 where G1: G1 =
    {nodes = G1-nodes, edges = {(n, e, n'). (n, e, n') ∈ edges (del-unEdge v w v' G)
                                ∧ n ∈ G1-nodes ∧ n' ∈ G1-nodes}}
    by metis
  obtain G2-nodes where G2-nodes: G2-nodes =
    {n. ∃ ps. valid-graph.is-path (del-unEdge v w v' G) n ps v'}
    by metis
  then obtain G2 where G2: G2 =
    {nodes = G2-nodes, edges = {(n, e, n'). (n, e, n') ∈ edges (del-unEdge v w v' G)
                                ∧ n ∈ G2-nodes ∧ n' ∈ 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-unMultigraph G1 using valid-unMultigraph-def by auto
  have valid-G2: valid-unMultigraph G2
using G2 valid-unMultigraph.corres[OF valid0'] valid-unMultigraph.no-id[OF valid0']

by (unfold-locales, auto)

hence valid-G2': valid-graph G2 using valid-unMultigraph-def by auto

have nodes G1={n. ∃ ps. valid-graph.is-path (del-unEdge v w v') n ps v}

using G1-nodes G1 by auto

moreover have edges G1={(n,e,n'). (n,e,n')∈edges (del-unEdge v w v')
 ∧ n∈nodes G1 ∧ n'∈nodes G1}

using G1-nodes G1 by auto

moreover have nodes G2={n. ∃ ps. valid-graph.is-path (del-unEdge v w v') n ps v'}

using G2-nodes G2 by auto

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

proof (rule ccontr)

assumes n∈nodes G1 ∪ nodes G2≠ nodes (del-unEdge v w v')

moreover have nodes G1 ⊆ nodes (del-unEdge v w v')

using valid-graph.is-path-memb[OF valid0] G1 G1-nodes by auto

moreover have nodes G2 ⊆ nodes (del-unEdge v w v')

using valid-graph.is-path-memb[OF valid0] G2 G2-nodes by auto

ultimately obtain n where n:

n∈nodes (del-unEdge v w v') n'∈nodes G1 n'∈nodes G2

by auto

hence n-neg-v : ¬(∃ ps. valid-graph.is-path (del-unEdge v w v') n ps v)

and

n-neg-v': ¬(∃ ps. valid-graph.is-path (del-unEdge v w v') n ps v')

using G1 G1-nodes G2 G2-nodes by auto

hence n≠v by (metis n(1) valid0 valid-graph.is-path-simps(1))

then obtain nvs where nvs: is-path n nvs v using (connected)

by (metis E-validD(1) assms(3) connected-def del-UnEdge-node n(1))

then obtain nvs' where nvs': nvs'=takeWhile (λx. x≠(v,w,v') ∧ x≠(v',w,v))

nvs by auto

moreover have nvs-nvs': nvs=nvs' @ dropWhile (λx. x≠(v,w,v') ∧ x≠(v',w,v))

nvs using nvs' takeWhile-dropWhile-id by auto

ultimately obtain n' where is-path-nvs': is-path n nvs' n'

and is-path n' (dropWhile (λx. x≠(v,w,v') ∧ x≠(v',w,v)) nvs) v

using nvs is-path-split[of n nvs' dropWhile (λx. x≠(v,w,v') ∧ x≠(v',w,v))]

nvs by auto

have n'=v ∨ n'=v'

proof (cases dropWhile (λx. x≠(v,w,v') ∧ x≠(v',w,v)) nvs)

case Nil

hence nvs=nvs' using nvs-nvs' by (metis append-Nil2)

hence n'=v using nvs is-path-nvs' path-end by (metis (mono-tags is-path.siraps(1)))

thus ?thesis by auto

next
case (Cons x xs)
hence dropWhile \((\lambda x. x \neq (v, w, v') \land x \neq (v', w, v))\) nvs \neq [] by auto
hence hd (dropWhile \((\lambda x. x \neq (v, w, v') \land x \neq (v', w, v))\) nvs) = (v, w, v)
  \lor hd (dropWhile \((\lambda x. x \neq (v, w, v') \land x \neq (v', w, v))\) nvs) = (v', w, v)
by (metis (lifting, full-types) hd-dropWhile)
hence x = (v, w, v') \lor x = (v', w, v) using Cons by auto
thus \(\lnot\)thesis
  using \(\langle\text{is-path } n' (\text{dropWhile } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)) \text{nvs})\rangle\)
  by (metis Cons is-path.simps(2))
qed
moreover have valid-graph.is-path (del-unEdge v w v' G) n nvs n'
  using is-path-nvs n' nvs'
proof (induct nvs' arbitrary:n nvs)
case Nil
thus \(\exists\)case by (metis del-UnEdge-node is-path.simps(1) valid0 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)
hence is-path x3 x n' using Cons by auto
moreover have xs = takeWhile \((\lambda x. x \neq (v, w, v') \land x \neq (v', w, v))\) (tl)
  using \(\langle x \neq xs = \text{takeWhile } (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v)) \text{nvs}\rangle\)
  by (metis (lifting, no-types) append-Cons list.distinct(1) takeWhile.simps(2)
      takeWhile-dropWhile-id list.sel(3))
ultimately have valid-graph.is-path (del-unEdge v w v' G) x3 xs n'
  using Cons by auto
moreover have x \neq (v, w, v') \land x \neq (v', w, v)
  using Cons(3) set-takeWhileD[of x (\lambda x. x \neq (v, w, v') \land x \neq (v', w, v))]
by (metis List.set.simps(2) insertI1)
hence x \in edges (del-unEdge v w v' G)
  by (metis Cons.prems(1) del-UnEdge-frame is-path.simps(2) x)
ultimately show \(\exists\)case using x
by (metis Cons.prems(1) is-path.simps(2) valid0 valid-graph.is-path.simps(2))
qed
ultimately show False using n-neg-v n-neg-v' by auto
qed
moreover have nodes G1 \cap nodes G2 = {}
proof (rule ccontr)
assume nodes G1 \cap nodes G2 \neq {}
then obtain n where n:n\in nodes G1 \land n \in nodes G2 by auto
then obtain nvs n' in\'s where
  nvs : valid-graph.is-path (del-unEdge v w v' G) n nvs 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 ((\text{rev-path nvs})@n'vs') v'
  using valid-unMultigraph.is-path-rev[OF valid0'] valid-graph.is-path-split[OF

39
valid0)
by auto
hence valid-unMultigraph.connected (del-unEdge v 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 v' G)

proof (rule ccontr)
assume edges 𝐺1 ∪ edges 𝐺2 ≠ edges (del-unEdge v v' G)
moreover have edges 𝐺1 ⊆ edges (del-unEdge v v' G) using 𝐺1 by auto
moreover have edges 𝐺2 ⊆ edges (del-unEdge v v' G) using 𝐺2 by auto
ultimately obtain n e n' where
  (n,e,n')∈edges (del-unEdge v v' G)
  (n,e,n')∉edges 𝐺1 (n,e,n')∉edges 𝐺2
by auto
moreover have n∈nodes (del-unEdge v v' G)
by (metis nen'(1) valid0 valid-graph.E-validD(1))
moreover have n'∈nodes (del-unEdge v 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 v' G)) by auto
moreover have n∈nodes 𝐺1 ⇒ n'∈nodes 𝐺2 ⇒ False

proof =
  assume n∈nodes 𝐺1 n'∈nodes 𝐺2
  then obtain nvs nv's where
    nvs : valid-graph.is-path (del-unEdge v v' G) n nvs v and
    nv's : valid-graph.is-path (del-unEdge v v' G) n' nv's v'
using 𝐺1 𝐺2 𝐺1-nodes 𝐺2-nodes by auto
hence valid-graph.is-path (del-unEdge v v' G) v
    ((rev-path nvs)@(n,e,n')#nv's) v'
using valid-unMultigraph.is-path-rev[OF valid0'] valid-graph.is-path-split'[OF valid0]

valid0)
  (n,e,n')∈edges (del-unEdge v v' G):
by auto
hence valid-unMultigraph.connected (del-unEdge v 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 nvs where
    n'vs : valid-graph.is-path (del-unEdge v v' G) n' n'vs v and
    nvs : valid-graph.is-path (del-unEdge v v' G) n nvs v'
using 𝐺1 𝐺2 𝐺1-nodes 𝐺2-nodes by auto
moreover have (n',e,n)∈edges (del-unEdge v 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') G1 n ps v 
using G1 G1-nodes n by auto 

hence G1 G1-nodes n by auto 

hence ps:valid-graph.is-path G1 n ps v 

proof (induct ps arbitrary:n)

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)

next 

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))
by (metis Cons.prems valid0 valid-graph.is-path.simps(2) x)
hence valid-graph.is-path G1 x3 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-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-unMultigraph.is-trail-intro)

qed
hence valid-graph.is-path G1 v (rev-path ps’) n’
using valid-unMultigraph.is-path-rev[OF valid-G1]
by auto
hence valid-graph.is-path G1 n (ps@(rev-path ps’)) n’
using ps valid-graph.is-path-split[OF valid-G1’,of n ps rev-path ps’ n’]
by auto
thus ∃ ps. valid-graph.is-path G1 n ps n’ by auto
qed
moreover have valid-unMultigraph.connected G2
unfolding valid-unMultigraph.connected-def[OF valid-G2]
proof (rule,rule,rule)
fix n n’
assume \( n : n \in \text{nodes } G \)
assume \( n' : n' \in \text{nodes } G \)
assume \( n \neq n' \)

obtain \( ps \) where \( \text{valid-graph.is-path } (\text{del-unEdge } v w v') n ps v' \)
using \( G \text{2-nodes } n \) by auto

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

proof (induct \( ps \) arbitrary:\( n \))
case Nil

moreover have \( v' \in \text{nodes } G \) using \( G \text{2-nodes } valid0 \)
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-G2 valid-unMultigraph.is-trail.simps(1)
valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)

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

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

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

hence \( \text{valid-graph.is-path } G2 \, x3 \, x' \) 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-G2' by (metis valid-graph.is-path.simps(2))

qed

obtain \( ps' \) where \( \text{valid-graph.is-path } (\text{del-unEdge } v w v') G n' ps' v' \)
using \( G \text{2-nodes } n' \) by auto

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

proof (induct \( ps' \) arbitrary:\( n' \))
case Nil

moreover have \( v' \in \text{nodes } G \) using \( G \text{2-nodes } valid0 \)
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-G2 valid-unMultigraph.is-trail.simps(1)
valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)

next
case (Cons \( x \, x s \))
obtain \( x1 \, x2 \, x3 \) where \( x : x = (x1, x2, x3) \) by (metis prod-cases3)

43
have \( x \in \text{nodes } G_2 \) 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 \((x_1, x_2, x_3) \in \text{edges } G_2\)
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 \( \langle \text{nodes } G_1 \cap \text{nodes } G_2 = \{\} \rangle \langle \text{edges } G_1 \cup \text{edges } G_2 = \text{edges } (\text{del-unEdge } v \ w \ v') G \rangle \)
by (metis IntI Un-iff assms(1) bex-empty connected-def del-UnEdge-node valid0 valid0'
valid-G1 valid-G2 valid-graph.E-validD(1) valid-graph.E-validD(2) valid-unMultigraph.no-id)
moreover have \( \text{valid-graph.is-path } (\text{del-unEdge } v \ w \ v') G \ x' x s v' \)
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 \( \langle \text{nodes } G_1 \cap \text{nodes } G_2 = \{\} \rangle \langle \text{edges } G_1 \cup \text{edges } G_2 = \text{edges } G \rangle \)
by (metis IntI Un-iff assms(1) bex-empty connected-def del-UnEdge-node valid0 valid0'
valid-G1 valid-G2 valid-graph.E-validD(1) valid-graph.E-validD(2) valid-unMultigraph.no-id)
moreover have \( \text{valid-graph.is-path } G_2 \ x' x s v' \)
by (metis Cons.prems valid0 valid-graph.is-path.simps(2) \( x \))
ultimately show \(?case\) using \( x \) valid-G2' by (metis valid-graph.is-path.simps(2))

qed

hence \( \text{valid-graph.is-path } G_2 \ v' (\text{rev-path } ps') n' \)
using \( \text{valid-unMultigraph.is-path-rev}[OF valid-G2]\)
by auto

hence \( \text{valid-graph.is-path } G_2 \ n \ (ps @ (\text{rev-path } ps')) n' \)
using \( \text{ps valid-graph.is-path-split}[OF valid-G2', of n ps rev-path ps' n'] \)
by auto
thus \( \exists ps. \text{valid-graph.is-path } G_2 \ n \ ps \ n' \) by auto

qed
ultimately show \(?thesis\) using valid-G1 valid-G2 that by auto

qed

lemma sub-graph-degree-frame:
 assumes \( \text{valid-graph } G_2 \ \text{edges } G_1 \cup \text{edges } G_2 = \text{edges } G \ \text{nodes } G_1 \cap \text{nodes } G_2 = \{\} \ n \in \text{nodes } G_1 \)
shows \( \text{degree } n G = \text{degree } n G_1 \)
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 \( \text{valid-graph.E-validD}[OF \ (\text{valid-graph } G_2)] \ \langle \text{fst } e = n \rangle \)
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 \( \{ e \in \text{edges } G \mid \text{fst } e = n \} \subseteq \{ e \in \text{edges } G. \text{fst } e = n \} \)
by (metis (lifting) Collect_mono Un_iff assms(2))
ultimately show \( \text{thesis} \) unfolding degree_def by auto

qed

lemma odd-nodes-no-edge[simp]: finite (nodes \( g \)) \( \Rightarrow \) num-of-odd-nodes \( g \) (\{ edges := \} | |) = 0

unfolding num-of-odd-nodes_def odd-nodes-set_def degree_def by simp

4 Adjacent nodes

definition (in valid-unMultigraph) adjacent:: \( 'v \Rightarrow 'v \Rightarrow \text{bool} \)
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 \( v \) \( v' \) \( \Rightarrow \) \( v \neq v' \)

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 \( E \) \( \Rightarrow \) finite \( \{ n. \ \text{adjacent } v \ n \} \)

proof –
assume \( \text{finite } E \)
\{ fix \( S \) \( v \)
have \( \text{finite } S \Rightarrow \text{finite } \{ n. \ \exists w. (v,w,n) \in S \} \)
proof (induct \( S \) rule: finite-induct)
case empty
thus \( ?\text{case} \) by auto
next
case \( (\text{insert } x \ F) \)
obtain \( x1 \) \( x2 \) \( x3 \) \( \text{where } x := (x1,x2,x3) \) by (metis prod_cases3)
have \( x1 = v \ \Rightarrow \ ?\text{case} \)
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 \( ?\text{thesis} \) using insert by auto
qed
moreover have \( x1 \neq v \ \Rightarrow \ ?\text{case} \)
proof –
assume \( x1 \neq v \)
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 

due

ultimately show \?thesis using \text{insert} by auto 

due } 

note \text{aux}=this 

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

due 

5 Undirected simple graph 

locale valid-unSimpGraph for G::(v,v,w) graph+ 

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


lemma (in valid-unSimpGraph) finV-to-finE[simp]: 

assumes finite V 

shows finite E 

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

case True 

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

thus finite E by auto 

due 

next 

case False 

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

moreover have finite (V \times V) using (finite V) by auto 

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

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

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

proof -- 

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

proof -- 

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

unfolding adjacent-def by auto 

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

unfolding adjacent-def by force 

ultimately show \?thesis by force 

due 

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

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

due 

ultimately show finite E by (metis card-infinite) 

due 

lemma del-unEdge-valid[simp]: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 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
q Ed

 lemma (in valid-unSimpGraph) del-UnEdge-non-adj:
  \((v, w, u) \in E \Rightarrow \neg \text{valid-unMultigraph.adjacent} (\text{del-unEdge} v w u G)\) v u
proof
  assume \((v, w, u) \in E\)
  and ccontr: valid-unSimpGraph-axioms (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 \((v, w', u) \in \text{edges} G\)
    using ccontr unfolding valid-unSimpGraph-axioms-def by auto
  hence \(w' \neq w\) by auto
  moreover have \((v, w', u) \in E\) using \(vw'\) unfolding del-unEdge-def by auto
  ultimately show False using no-multi[of v w u v]
q Ed

 lemma (in valid-unSimpGraph) degree-adjacent: finite \(E \Rightarrow \text{degree} v G = \text{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-axioms-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
q Ed
  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 G valid-unSimpGraph-def
q Ed

 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 = degree v (del-unEdge v w u G)

using Suc n = degree v G: (v, w, u) ∈ 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 degree v (del-unEdge v w u G)

= card (Collect (valid-unMultigraph.adjacent (del-unEdge v w u G) v))

using Suc.hyps by auto

moreover have Suc(card (\{n. valid-unMultigraph.adjacent (del-unEdge v w u G) v n\})) = card (\{n. valid-unMultigraph.adjacent G v n\})

using valid-unMultigraph.adjacent-def[OF valid]

proof –

have \{n. valid-unMultigraph.adjacent (del-unEdge v w u G) v n\} ⊆ \{n. valid-unMultigraph.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\∈\{n. valid-unMultigraph.adjacent G v n\}

using (v, w, u)\∈ edges G: unfolding valid-unMultigraph.adjacent-def[of valid]

by auto

ultimately have \{n. valid-unMultigraph.adjacent (del-unEdge v w u G) v n\} \∪ \{u\}

⊆ \{n. valid-unMultigraph.adjacent G v n\} by auto

moreover have \{n. valid-unMultigraph.adjacent G v n\} - \{u\}

⊆ \{n. valid-unMultigraph.adjacent (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. valid-unMultigraph.adjacent (del-unEdge v w u G) v n\} \∪ \{u\}

= \{n. valid-unMultigraph.adjacent G v n\} by auto

moreover have u\∉\{n. valid-unMultigraph.adjacent (del-unEdge v w u G) v n\}

using valid-unSimpGraph.del-UnEdge-non-adj[of valid-unSimpGraph G]

\{(v, w, u)\∈ edges G\}

by auto

moreover have finite \{n. valid-unMultigraph.adjacent G v n\}

using valid-unMultigraph.adjacent-finite[of valid finite (edges G)] by simp

ultimately show ?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

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

6 Definition of 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 (\text{rem-unPath} (rev-path ps) G) = edges (\text{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 \( \forall v \in V. \text{ even (degree} v G) \)
proof -
obtain v ps v' where cycle:is-Eulerian-circuit v ps v' using assms by auto
hence edges (\text{rem-unPath} ps G) = \{\}
  unfolding is-Eulerian-circuit-def is-Eulerian-trail-def
  by simp
moreover have nodes (\text{rem-unPath} ps G) = nodes G by auto
ultimately have rem-unPath ps G = G (\{edges:=\{\}) by auto
hence num-of-odd-nodes (\text{rem-unPath} ps G) = 0 by (metis assms(2) odd-nodes-no-edge)
moreover have v=v'

7 Necessary conditions for Eulerian trails and circuits

theory KoenigsbergBridge imports MoreGraph Map Enum
begin


by \((\text{metis \ is-Eulerian-circuit \ v \ ps \ v' is-Eulerian-circuit-def})\)
hence \(\text{num-of-odd-nodes} (\text{rem-unPath \ ps \ G}) = \text{num-of-odd-nodes} \ G\)
by \((\text{metis \ assms(2) \ assms(3) \ cycle \ is-Eulerian-circuit-def})\)
\text{is-Eulerian-trail-def \ rem-UnPath-cycle})\)
ultimately have \(\text{num-of-odd-nodes} \ G = 0\) by auto
moreover have \(\text{finite} (\text{odd-nodes-set} \ G)\)
using \((\text{finite} \ V) \ \text{unfolding} \ \text{odd-nodes-set-def} \ \text{by auto}\)
ultimately have \(\text{odd-nodes-set} \ G = {}\) unfolding \(\text{num-of-odd-nodes-def}\) by auto
thus \(?\text{thesis} \ \text{unfolding} \ \text{odd-nodes-set-def} \ \text{by auto}\)
qed

\text{theorem (in valid-unMultigraph) \ euclerian-path-ex:}
assumes \((\forall \ v \in V. \ \text{even} (\text{degree} \ v \ G)) \lor (\text{num-of-odd-nodes} \ G = 2)\)
shows \((\forall v \in V. \ \text{even} (\text{degree} \ v \ G)) \lor (\text{num-of-odd-nodes} \ G = 2)\)
proof
  obtain \(v \ ps \ v'\) where \(\text{path: \ is-Eulerian-trail \ v \ ps \ v'}\)
  using \(\text{assms by auto}\)
hence \(\text{edges} (\text{rem-unPath \ ps \ G}) = {}\)
  unfolding \(\text{is-Eulerian-trail-def}\)
  by simp
moreover have \(\text{nodes} (\text{rem-unPath \ ps \ G}) = \text{nodes} \ G\) by auto
ultimately have \(\text{rem-unPath \ ps \ G} = G \ \text{edges:=\{}\}\) by auto
hence \(\text{odd-nodes: \ num-of-odd-nodes} (\text{rem-unPath \ ps \ G}) = 0\)
by \((\text{metis \ assms(2) \ odd-nodes-no-edge})\)
have \(v \neq v' \implies ?\text{thesis}\)
proof \(\text{cases even(\text{degree} \ v' \ G)}\)
case True
  assume \(v \neq v'\)
  have \(\text{is-trail \ v \ ps \ v'}\) by \((\text{metis \ is-Eulerian-trail-def \ path})\)
hence \(\text{num-of-odd-nodes} (\text{rem-unPath \ ps \ G}) = \text{num-of-odd-nodes} \ G\)
  + (if even (\text{degree} \ v \ G) then 2 else 0)
  using \(\text{rem-UnPath-even \ true} \ \text{finite} \ V \ \text{finite} \ E \ \text{v \neq v'} \ \text{by auto}\)
hence \(\text{num-of-odd-nodes} \ G + (\text{if even (degree} \ v \ G) \ \text{then} \ 2 \ \text{else 0}) = 0\)
  using \(\text{odd-nodes by auto}\)
hence \(\text{num-of-odd-nodes} \ G = 0\) by auto
moreover have \(\text{finite} (\text{odd-nodes-set} \ G)\)
using \((\text{finite} \ V) \ \text{unfolding} \ \text{odd-nodes-set-def} \ \text{by auto}\)
ultimately have \(\text{odd-nodes-set} \ G = {}\) unfolding \(\text{num-of-odd-nodes-def}\) by auto
thus \(?\text{thesis} \ \text{unfolding} \ \text{odd-nodes-set-def} \ \text{by auto}\)
next
case False
  assume \(v \neq v'\)
  have \(\text{is-trail \ v \ ps \ v'}\) by \((\text{metis \ is-Eulerian-trail-def} \ \text{path})\)
hence \(\text{num-of-odd-nodes} (\text{rem-unPath \ ps \ G}) = \text{num-of-odd-nodes} \ G\)
  + (if odd (\text{degree} \ v \ G) then \(-2\) else 0)
  using \(\text{rem-UnPath-odd \ false} \ \text{finite} \ V \ \text{finite} \ E \ :v \neq v' \ \text{by auto}\)
hence \(\text{odd-nodes-if} : \text{num-of-odd-nodes} \ G + (\text{if odd (degree} \ v \ G) \ \text{then} \ -2 \ \text{else
0) = 0
using odd-nodes by auto

have odd (degree v G) ⇒ thesis
proof
  assume odd (degree v G)
  hence num-of-odd-nodes G = 2 using odd-nodes-if by auto
  thus thesis by simp
qed

moreover have even (degree v G) ⇒ 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 thesis unfolding odd-nodes-set-def by auto
qed

ultimately show thesis by auto
qed

moreover have v = v' ⇒ thesis
by (metis assms (2) assms (3) eulerian-cycle-ex is-Eulerian-circuit-def path)
ultimately show 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 ≡ P a ∧ P b ∧ P c ∧ P d
definition [simp]: enum-class.enum-ex P ≡ P a ∨ P b ∨ P c ∨ P d
instance proof qed (auto, (case-tac x, auto)+)
end
instantiation kon-bridge :: enum

begin

definition [simp]:enum-class.enum = [ab1, ab2, ac1, ac2, ad1, cd1, bd1]

definition [simp]:enum-class.enum-all P ⟷ P ab1 ∧ P ab2 ∧ P ac1 ∧ P ac2
∧ P ad1 ∧ P bd1

definition [simp]:enum-class.enum-ex P ⟷ P ab1 ∨ P ab2 ∨ P ac1 ∨ P ac2
∨ P ad1 ∨ P bd1

instance proof qed (auto, (case-tac x, auto+)

end

interpretation kon-graph: valid-unMultigraph kon-graph

proof (unfold-locales)

show fst ' edges kon-graph ⊆ nodes kon-graph by eval

next

show snd ' snd ' edges kon-graph ⊆ nodes kon-graph by eval

next

have ∀ v w u'. ((v, w, u') ∈ edges kon-graph) = ((u', w, v) ∈ edges kon-graph)

by eval

thus ∀ v w u'. ((v, w, u') ∈ edges kon-graph) = ((u', w, v) ∈ edges kon-graph)

by simp

next

have ∀ v w. (v, w, v) /∈ edges kon-graph by eval

thus ∀ v w. (v, w, v) /∈ edges kon-graph by simp

qed

theorem ¬kon-graph.is-Eulerian-trail v1 p v2

proof

assume kon-graph.is-Eulerian-trail v1 p v2

moreover have finite (nodes kon-graph) by (metis finite-code)

moreover have finite (edges kon-graph) by (metis finite-code)

ultimately have contra:
(∀ v ∈ nodes kon-graph. even (degree v kon-graph)) ∨ (num-of-odd-nodes kon-graph = 2)

by (metis kon-graph.euclerian-path-ex)

have odd(degree a kon-graph) by eval

moreover have odd(degree b kon-graph) by eval

moreover have odd(degree c kon-graph) by eval

moreover have odd(degree d kon-graph) by eval

ultimately have ¬(num-of-odd-nodes kon-graph = 2) by eval

moreover have ¬(∀ v ∈ nodes kon-graph. even (degree v kon-graph)) by eval

ultimately show False using contra by auto

qed
9 Sufficient conditions for Eulerian trails and circuits

Lemma (in valid-unMultigraph) eulerian-cons:

assumes
valid-unMultigraph.is-Eulerian-trail (del-unEdge v0 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 v1 G)
  using valid-unMultigraph-axioms by auto
hence distinct:valid-unMultigraph.is-trail (del-unEdge v0 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 v1 G)
  using valid-unMultigraph.path-in-edges[OF valid] by auto
moreover have (v0,w,v1)∈ edges (del-unEdge v0 v1 G)
  unfolding del-unEdge-def by auto
moreover have (v1,w,v0)∈ edges (del-unEdge v0 v1 G)
  unfolding del-unEdge-def by auto
ultimately have (v0,w,v1)∈ set ps (v1,w,v0)∈ set ps 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 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 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 v2 G)
  using valid-unMultigraph-axioms del-unEdge-valid by auto
have del-unEdge v2 v3 G =del-unEdge v3 v2 G
  by (metis delete-edge-sym)

hence valid-unMultigraph.is-Eulerian-trail (del-unEdge v3 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 eulerian-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) v2']
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' v e. (v, e, v') ∈ edges G1 ⇒ v' ∈ nodes G1))

54
next

fix aa ab ba

assume ba ∉ nodes G2 (aa, ab, ba) ∈ edges G2
thus ba ∈ 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 @(((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 ⊆ edges G by (metis G UnI1 Un-assoc select-convs(2) subrelI)

moreover have nodes G1 ⊆ 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 ⊆ edges G by (metis G inf-sup-ord(3) le-supE select-convs(2))

moreover have nodes G2 ⊆ nodes G by (metis G inf-sup-ord(4) select-convs(1))

ultimately show ?thesis
using distinct-path-subset[of G2 G, OF valid-unMultigraph G2: valid]

by auto

qed

have valid-graph.is-path G v1 (ps1 @(((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) ∈ edges G

using G by auto

ultimately show ?thesis
using valid-graph.is-path-split[OF valid’, of v1 ps1 v1’ w v2 ps2 v2’] by auto

qed

moreover have distinct (ps1 @(((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
v1 ?
valid-unMultigraph.path-in-edges[OF valid-unMultigraph G2], of v2 ps2
v2 ?
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 set ps2 ∩ set (rev-path ps1) = {}
  by (metis calculation(2) distinct-append distinct-rev path ps1-G ps2-G
rev-path-append
  rev-path-double valid valid-unMultigraph.is-trail-path)
moreover have (v2,w,v1') ∉ set (ps1@((v1',w,v2)#ps2))
proof –
  have (v2,w,v1') ∉ edges G1
    using v2 ∈ nodes G2 { valid-graph G1
    by (metis Int-Iff all-not-in-conv assms(1) valid-graph. E-validD(1))
  hence (v2,w,v1') ∈ set ps1
  by (metis assms(3) assms(5) split-list valid-unMultigraph.is-trail-split')
valid-unMultigraph.is-Eulerian-trail-def)
moreover have (v2,w,v1') ∉ edges G2
  using v1' ∈ nodes G1 { valid-graph G2
  by (metis Int-Iff assms(1) empty-Iff valid-graph. E-validD(2))
  hence (v2,w,v1') ∉ 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 (v2,w,v1') ≠ (v1',w,v2)
  using v1' ∈ nodes G1 \ v2 ∈ nodes G2
  by (metis Int-Iff 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 edges (rem-unpath (ps1@((v1',w,v2)#ps2)) G) = {}
proof –
  have edges (rem-unPath (ps1@((v1',w,v2)#ps2)) G) = edges G –
(set ps1@((v1',w,v2)#ps2)) ∪ set (rev-path (ps1@((v1',w,v2)#ps2)))
  by (metis rem-unPath-edges)
  also have ... = edges G – (set ps1 ∪ set ps2 ∪ set (rev-path ps1) ∪ set (rev-path ps2)
∪ {(v1',w,v2),(v2,w,v1')}) using rev-path-append by auto
finally have edges (rem-unPath (ps1@((v1',w,v2)#ps2)) G) = edges G –
(set ps1 ∪
set ps2 ∪ set (rev-path ps1) ∪ set (rev-path ps2) ∪ {(v1',w,v2),(v2,w,v1')})

moreover have edges (rem-unPath ps1 G1) = {}
  by (metis assms(3) valid-unMultigraph.is-Eulerian-trail-def)
hence edges G1 = (set ps1 ∪ set (rev-path ps1)) = {}
  by (metis rem-unPath-edges)
moreover have edges (rem-unPath ps2 G2) = {}
  by (metis assms(4) assms(6) valid-unMultigraph.is-Eulerian-trail-def)
hence edges G2 = (set ps2 ∪ set (rev-path ps2)) = {}

57
ultimately show \(\text{thesis}\) using \(G\) by auto

q.e.d.

\textbf{lemma} (in \textit{valid-unMultigraph}) eulerian-sufficient:
\begin{itemize}
  \item \textbf{assumes} finite \(V\) finite \(E\) connected \(V \neq \{\}\)
  \item \textbf{shows} num-of-odd-nodes \(G = 2 \implies\)
    \((\exists v \in V. \exists v' \in V. \exists ps. \text{odd}(\text{degree } v \ G) \land \text{odd}(\text{degree } v' \ G) \land \text{is-Eulerian-trail } v \ ps \ v')\)
    \text{and} num-of-odd-nodes \(G = 0 \implies\) \((\forall v \in V. \exists ps. \text{is-Eulerian-circuit } v \ ps \ v)\)
  \item \textbf{using} \(\text{finite } E\) \(\text{finite } V\) \text{valid-unMultigraph-axioms} \(\langle V \neq \{\}\rangle\) \(\text{connected}\)
  \item \textbf{proof} (induct \(E\) arbitrary; \(G\) rule: \text{less-induct})
  \item \textbf{case} \(\text{less}\)
    \item \textbf{assume} finite \((\text{edges } G)\) \text{and} finite \((\text{nodes } G)\) \text{and} \text{valid-unMultigraph } G \text{ and}
    \item nodes \(G \neq \{\}\)
    \item and \text{valid-unMultigraph.\textunderscore connected} \text{ and} num-of-odd-nodes \(G = 2\)
  \item \textbf{have} valid-graph \(G\) \text{using} \(\text{valid-unMultigraph } G\) \text{valid-unMultigraph-def} by auto
  \item \textbf{obtain} \(n_1\) \(n_2\) where
    \item \(n_1\) \(\in\) nodes \(G\) \text{odd}(\text{degree } n_1 \ G)
    \item and \(n_2\) \(\in\) nodes \(G\) \text{odd}(\text{degree } n_2 \ G)
    \item and \(n_1 \neq n_2\) unfolding num-of-odd-nodes-def odd-nodes-set-def
  \item \textbf{proof} --
    \item have \(\forall S. \text{card } S = 2 \implies (\exists n_1\ n_2. n_1 \in S \land n_2 \in S \land n_1 \neq n_2)\)
      \text{by} (metis \text{card-\textbullet-0-iff} \text{equals\textbullet0} \text{even-card\textbullet} \text{even-numeral} \text{zero-neg-numeral})
    \item then obtain \(t_1\) \(t_2\)
      \item where \(t_1 \in \{ v \in \text{nodes } G. \text{odd } (\text{degree } v \ G)\} t_2 \in \{ v \in \text{nodes } G. \text{odd } (\text{degree } v \ G)\} t_1 \neq t_2\)
        \text{using} \(\text{num-of-odd-nodes } G = 2\) unfolding num-of-odd-nodes-def odd-nodes-set-def
        \text{by} force
        \item thus \(\text{thesis}\) by (metis (lifting) that \text{mem-Collect-eq})
  \item q.e.d.
    \item \textbf{have} even-\textbullet-\textbullet-\textbullet-\textbullet-\textbullet-\textbullet\(\land n. n \in \text{nodes } G \implies n \neq n_1 \implies n \neq n_2 \implies \text{even}(\text{degree } n \ G)\)
      \text{proof} (rule \text{ccontr})
      \item fix \(n\) assume \(n \in \text{nodes } G\) \(n \neq n_1\) \(n \neq n_2\) \text{odd } (\text{degree } n \ G)
      \item have \(n \in \text{odd-nodes-set } G\)
        \text{by} (metis \text{mono-tags} \(\forall n. n \in \text{nodes } G. \text{odd } (\text{degree } n \ G) \implies \text{mem-Collect-eq odd-nodes-set-def}\))
      \item moreover have \(n_1 \in \text{odd-nodes-set } G\)
        \text{by} (metis \text{mono-tags} \text{mem-Collect-eq } n_1(1) n_1(2) \text{odd-nodes-set-def})
      \item moreover have \(n_2 \in \text{odd-nodes-set } G\)
        \text{using} \(n_2(1) n_2(2)\) unfolding odd-nodes-set-def by auto
      \item ultimately have \(\{ n, n_1, n_2 \} \subseteq \text{odd-nodes-set } G\) by auto
      \item moreover have \(\text{card}\{ n, n_1, n_2 \} \geq 3\) using \(n_1 \neq n_2\) \(n \neq n_1\) \(n \neq n_2\) by auto
      \item moreover have finite \(\text{odd-nodes-set } G\)
        \text{using} \(\text{finite } (\text{nodes } G)\) unfolding odd-nodes-set-def by auto
      \item ultimately have \(\text{card } (\text{odd-nodes-set } G) \geq 3\)
using card-mono[of odd-nodes-set G \( \{n_1, n_2\} \)] by auto

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

qed

have \( \{ e \in \text{edges } G. \text{fst } e = n_1 \} \neq \{ \} \)
  using n1
by (metis (full-types) degree-def empty-iff finite.emptyI odd-card)

then obtain \( v' \) where \( (n_1, w, v') \in \text{edges } G \) by auto

have \( v' = n_2 \implies (\exists \; v' \in \text{nodes } G. \exists \; ps. \; \text{odd } (\text{degree } v G) \land \text{odd } (\text{degree } v' G) \land v \neq v' \)
  \land \text{valid-unMultigraph.is-Eulerian-trail } G \; v \; ps \; v' \)

proof (cases valid-unMultigraph.connected (del-unEdge n1 w n2 G))
  assume \( v' = n_2 \)
  assume connected':valid-unMultigraph.connected (del-unEdge n1 w n2 G)
moreover have \( \text{num-of-odd-nodes } (\text{del-unEdge } n_1 \; w \; n_2 \; G) = 0 \)
  using \( ((n_1, w, v') \in \text{edges } G) \; (\text{finite } (\text{edges } G)) \; (\text{finite } (\text{nodes } G)) \; (v' = n_2) \)
(num-of-odd-nodes G = 2) \; (valid-unMultigraph G) \; (\text{del-UnEdge-odd-odd } n_1(2) \; n_2(2) \)
  by force
moreover have \( \text{finite } (\text{edges } (\text{del-unEdge } n_1 \; w \; n_2 \; G)) \)
  using \( (\text{finite } (\text{edges } G)) \) by auto
moreover have \( \text{finite } (\text{nodes } (\text{del-unEdge } n_1 \; w \; n_2 \; G)) \)
  using \( (\text{finite } (\text{nodes } G)) \) by auto
moreover have \( \text{edges } G - \{(n_1, w, n_2), (n_2, w, n_1)\} \subset \text{edges } G \)
  using Diff-iff Diff-subset \( ((n_1, w, v') \in \text{edges } G) \; (v' = n_2) \)
  by fast
hence \( \text{card } (\text{edges } (\text{del-unEdge } n_1 \; w \; n_2 \; G)) < \text{card } (\text{edges } G) \)
  using \( (\text{finite } (\text{edges } G)) \; (\text{psubset-card-mono[of edges } G \; \text{edges } G = \{(n_1,w,n_2),(n_2,w,n_1)\}) \)

unfolding del-unEdge-def by auto
moreover have \( \text{valid-unMultigraph } (\text{del-unEdge } n_1 \; w \; n_2 \; G) \)
  using \( (\text{valid-unMultigraph } G) \; (\text{del-unEdge-valid} \) by auto
moreover have \( \text{nodes } (\text{del-unEdge } n_1 \; w \; n_2 \; G) \neq \{ \} \)
  by (metis (full-types) del-UnEdge-node empty-iff n1(1))
ultimately have \( \forall \; v \in \text{nodes } (\text{del-unEdge } n_1 \; w \; n_2 \; G). \exists \; ps. \; \text{valid-unMultigraph.is-Eulerian-circuit } (\text{del-unEdge } n_1 \; w \; n_2 \; G) \; v \; ps \; v \)
  using less.hyps[of del-unEdge n1 w n2 G] by auto
thus ?thesis using eulerian-cons
  by (metis \( (n_1, w, v') \in \text{edges } G \) \( n_1 \neq n_2 \) \( v' = n_2 \) \( \text{valid-unMultigraph } G \))
  \( \text{valid-unMultigraph } (\text{del-unEdge } n_1 \; w \; n_2 \; G) \; \text{del-UnEdge-node } n_1(1) \; n_1(2) \)
\( n_2(1) \; n_2(2) \)
  \( \text{valid-unMultigraph.eulerian-cons valid-unMultigraph.is-Eulerian-circuit-def} \)

next
  assume \( v' = n_2 \)
  assume not-connected:-\text{valid-unMultigraph.connected } (\text{del-unEdge } n_1 \; w \; n_2 \; G)
  have \( \text{valid0:valid-unMultigraph } (\text{del-unEdge } n_1 \; w \; n_2 \; G) \)

59
using ⟨valid-unMultigraph G; del-unEdge-valid by auto

hence validG;valid-graph (del-unEdge n1 w n2 G)

using valid-unMultigraph-def by auto

have all-even∀ n∈nodes (del-unEdge n1 w n2 G). even(degree n (del-unEdge n1 w n2 G))

proof –

have even (degree n1 (del-unEdge n1 w n2 G))

using ⟨(n1, w, v') ∈ edges G; finite (edges G); v' = n2⟩ ⟨valid-unMultigraph G⟩ n1

by (auto simp add: valid-unMultigraph.corres)

moreover have even (degree n2 (del-unEdge n1 w n2 G))

using ⟨(n1, w, v') ∈ edges G; finite (edges G); v' = n2⟩ ⟨valid-unMultigraph G⟩ n2

by (auto simp add: valid-unMultigraph.corres)

moreover have ∀ n. n ∈ nodes (del-unEdge n1 w n2 G) ⇒ n ≠ n1 ⇒

n ≠ n2 ⇒

even (degree n (del-unEdge n1 w n2 G))

using valid-unMultigraph.degree-frame[OF ⟨valid-unMultigraph G; of - n1 n2 w] even-except-two

by (metis (no-types) finite (edges G); del-unEdge-def empty-iff insert-iff

select-convs(1))

ultimately show ?thesis by auto

qed

have (n1,w,n2)∈edges G by (metis ⟨(n1, w, v') ∈ edges G; v' = n2⟩)

hence (n2,w,n1)∈edges G by (metis valid-unMultigraph G valid-unMultigraph.corres)

obtain G1 G2 where

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

n ps n1}

and G1-edges: edges G1={(n,e,n'). (n,e,n')∈edges (del-unEdge n1 w n2 G)

∧ n∈nodes G1 ∧ n'∈nodes G1}

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

and G2-edges:edges G2={(n,e,n'). (n,e,n')∈edges (del-unEdge n1 w n2 G)

∧ n∈nodes G2 ∧ n'∈nodes G2}

and G1-G2-edges-union:edges G1 ∪ edges G2 = edges (del-unEdge n1 w n2 G)

and edges G1 ∩ edges G2={}

and G1-G2-nodes-union:nodes G1 ∪ nodes G2=nodes (del-unEdge n1 w n2 G)

and nodes G1 ∩ 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 ⟨valid-unMultigraph G

⟨valid-unMultigraph.connected G; ¬ valid-unMultigraph.connected (del-unEdge
\[ (n_1, w, n_2) \in \text{edges } G \].

have edges \((\text{del-unEdge } n_1 w n_2 G) \subset \text{edges } G\)

unfolding del-unEdge-def using \((n_1, w, n_2) \in \text{edges } G\), \((n_2, w, n_1) \in \text{edges } G\)

by auto

hence card \((\text{edges } G_1) < \text{card } (\text{edges } G)\) using G1-G2-edges-union

by (metis (full-types) finite (edges G) inf-sup-absorb less-infI2 psubset-card-mono)

moreover have finite (edges G1)

using G1-G2-edges-union

by (metis finite-Un less-imp-le rev-finite-subset)

moreover have nodes G1 \(\subseteq\) nodes \((\text{del-unEdge } n_1 w n_2 G)\)

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 \(n_1 \in \text{nodes } G_1\)

proof

have \(n_1 \in \text{nodes } (\text{del-unEdge } n_1 w n_2 G)\)

using \(n_1 \in \text{nodes } G\) by auto

hence \(\forall n \in \text{nodes } G_1. \text{degree } n G_1 = \text{degree } n \text{(del-unEdge } n_1 w n_2 G)\)

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

by (metis G1-G2-edges-union (nodes G1 \(\cap\) nodes G2 = \{\}))

hence \(\forall n \in \text{nodes } G_1. \text{even(degree } n G_1)\)

using all-even

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

thus \(\text{thesis using } G_1\)-nodes by auto

qed

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

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

proof

have valid-graph G2 using \(\text{valid-unMultigraph } G_2; \text{valid-unMultigraph-def}\)

by auto

hence \(\forall n \in \text{nodes } G_1. \text{degree } n G_1 = \text{degree } n (\text{del-unEdge } n_1 w n_2 G)\)

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

by (metis G1-G2-edges-union (nodes G1 \(\cap\) nodes G2 = \{\}))

hence \(\forall n \in \text{nodes } G_1. \text{even(degree } n G_1)\)

using all-even

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

thus \(\text{thesis using } G_1\)-nodes by auto

qed

ultimately have \(\forall v \in \text{nodes } G_1. \exists ps. \text{valid-unMultigraph.is-Eulerian-circuit } G_1 v ps v\)

using less.hyps[of G1] \(\text{valid-unMultigraph } G_1; \text{valid-unMultigraph.connected } G_1\)

by auto

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

using \(n_1 \in \text{nodes } G_1\)

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

have card \((\text{edges } G_2) < \text{card } (\text{edges } G)\)

using G1-G2-edges-union \((\text{edges } (\text{del-unEdge } n_1 w n_2 G) \subset \text{edges } G)\)

by (metis (full-types) finite (edges G) inf-sup-ord le-less-trans psubset-card-mono)
moreover have finite (edges G2)
using G1-G2-edges-union (finite (edges G))
by (metis edges (del-unEdge n1 w n2 G) ⊂ edges G; finite-Un less-imp-le
rev-finite-subset)
moreover have nodes G2 ⊆ nodes (del-unEdge n1 w n2 G)
by (metis G1-G2-nodes-union Un-upper2)
hence finite (nodes G2)
using (finite (nodes G)) del-UnEdge-node rev-finite-subset
by auto
moreover have n2 ∈ nodes G2
proof –
  have n2 ∈ nodes (del-unEdge n1 w n2 G)
  using (n2 ∈ nodes G) by auto
  hence valid-graph.is-path (del-unEdge n1 w n2 G) n2 [] n2
  using valid0’ by (metis valid-graph.is-path-simps(1))
  thus ?thesis using G2-nodes by auto
qed
hence nodes G2 ≠ {} by auto
moreover have num-of-odd-nodes G2 = 0
proof –
  have valid-graph G1 using (valid-unMultigraph G1) valid-unMultigraph-def
  by auto
  hence ∀ n ∈ nodes G2. degree n G2 = degree n (del-unEdge n1 w n2 G)
  using sub-graph-degree-frame[of G1 G2 (del-unEdge n1 w n2 G)]
  by (metis G1-G2-edges-union (nodes G1 ∩ nodes G2) = {}; inf-commute
  sup-commute)
  hence ∀ n ∈ nodes G2. even(degree n G2) using all-even
  by (metis G1-G2-nodes-union Un-iff)
  thus ?thesis
  unfolding num-of-odd-nodes-def odd-nodes-set-def
  by (metis (lifting) Collect-empty-eq card-eq-0-iff)
qed
ultimately have ∀ v ∈ nodes G2. ∃ ps. valid-unMultigraph.is-Eulerian-circuit
G2 v ps v
using less.hyps[of G2] (valid-unMultigraph G2) (valid-unMultigraph.connected G2)
by auto
then obtain ps2 where ps2:valid-unMultigraph.is-Eulerian-trail G2 n2 ps2 n2
using (n2 ∈ nodes G2)
by (metis (full-types) valid-unMultigraph G2; valid-unMultigraph.is-Eulerian-circuit-def)
have (nodes = nodes G1 ∪ nodes G2, edges = edges G1 ∪ edges G2 ∪ {(n1, w, n2),
(n2, w, n1)}) = G
proof –
  have edges (del-unEdge n1 w n2 G) ∪ {(n1, w, n2),(n2, w, n1)} = edges G
  using (n1, w, n2) ∈ edges G; (n2, w, n1) ∈ edges G
  unfolding del-unEdge-def by auto
moreover have nodes (del-unEdge n1 w n2 G) = nodes G

62
unrolling 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 n1 n2 ps2 n2 w]
by (metis (edges G1 \cap \text{edges } G2 = \{\} \} \{\text{nodes} G1 \cap \text{nodes } G2 = \{\} \} \langle\text{valid-unMultigraph } G1\rangle)
\langle\text{valid-unMultigraph } G2; \text{ps } ps2\rangle
ultimately show \(?\)thesis by (metis \(\langle n1 \neq n2\rangle \\langle n1(1) \ n2(1) \ n2(2)\rangle\))
qed
moreover have \(v' \neq n2 \rightarrow (\exists v \epsilon \text{nodes } G. \exists v' \epsilon \text{nodes } G. \exists \text{ps. odd } (\text{degree } v \ G) \land \text{odd } (\text{degree } v' \ G))\)
\(\land v \neq v' \land \text{valid-unMultigraph.is-Eulerian-trail } G \ v \ ps \ v'\)
proof (cases valid-unMultigraph.connected (\text{del-unEdge } n1 \ w \ v' \ G))
case True
assume \(v' \neq n2\)
assume connected':valid-unMultigraph.connected (\text{del-unEdge } n1 \ w \ v' \ G)
have \(n1 \in \text{nodes } (\text{del-unEdge } n1 \ w \ v' \ G)\) by (metis del-UnEdge-node n1(1))
hence \(even-n1\):even(\text{degree } n1 (\text{del-unEdge } n1 \ w \ v' \ G))
using valid-unMultigraph.del-UnEdge-even\langle OF \ (\text{valid-unMultigraph } G) \ (\langle n1, w, v'\rangle \in \text{edges } G)\rangle
\langle\text{finite } (\text{edges } G)\rangle\) \langle\text{odd } (\text{degree } n1 \ G)\rangle
unrolling odd-nodes-set-def by auto
moreover have \(\text{odd-n2}\) odd(\text{degree } n2 (\text{del-unEdge } n1 \ w \ v' \ G))
using valid-unMultigraph.degree-frame\langle OF \ (\text{valid-unMultigraph } G) \ (\langle n1, w, v'\rangle \in \text{edges } G)\rangle
\langle\text{finite } (\text{edges } G)\rangle\)
unrolling odd-nodes-set-def by auto
ultimately have two-odds::num-of-odd-nodes (del-unEdge n1 w v' G) = 2
  by (metis (lifting) v' \neq n2) \valid graph G \valid-unMultigraph G
  ν(n1, w, v') \in 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)\} \subset edges G
  using \{(n1, w, v'), w, v'\} \in 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))
  \unfolding del-unEdge-def by (metis \finite (nodes G) select-convs(1))

moreover have nodes (del-unEdge n1 w v' G) \neq \{\}
  by (metis (full-types) del-UnEdge-node empty-iff n1(1))

ultimately obtain s t ps where
  s: \subset nodes (del-unEdge n1 w v' G) odd (degree s (del-unEdge n1 w v' G))
  and t:t\in nodes (del-unEdge n1 w v' G) odd (degree t (del-unEdge n1 w v' G))

  G)) and s \neq t
  and s-ps-t: \valid-unMultigraph.is-Eulerian-trail (del-unEdge n1 w v' G) s t ps

using connected' less-hyps[of (del-unEdge n1 w v' G)] by auto

hence (s=n2 \land t=v') v (s=v' \land 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 \implies t=v' \implies ?thesis
  by (metis (n1, w, v') \in edges G \n1 \neq 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' \implies t=n2 \implies ?thesis
  by (metis (n1, w, v') \in edges G \n1 \neq n2 \valid-unMultigraph G \n1(1)
  n1(2) n2(1) n2(2)
  s-ps-t valid-unMultigraph.eulerian-cons)

ultimately show ?thesis by auto

next
case False

assume v' \neq n2

assume not-connected::valid-unMultigraph.connected (del-unEdge n1 w v' G)

have (v', w, n1) \in edges G using (n1, w, v') \in 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 \(\text{valid0};\text{valid-graph} (\text{del-unEdge} n1 w v' G)\)
using valid-unMultigraph-def by auto
have even-n1:even(degree n1 \(\text{del-unEdge} n1 w v' G\))
using valid-unMultigraph.del-UnEdge-even[OF valid-unMultigraph G]
\((n1,w,v')\in \text{edges} G\)
\(\exists \text{finite} (\text{edges} G)\) \(\forall n1\)
unfolding odd-nodes-set-def by auto
moreover have odd-n2:odd(degree n2 \(\text{del-unEdge} n1 w v' G\))
using \((n1 \neq n2) \land (v' \neq n2)\) valid-unMultigraph.degree-frame[OF valid-unMultigraph G]
\(\exists \text{finite} (\text{edges} G), \text{of} n2 n1 v' w\]
by auto
moreover have \(v' \neq n1\)
using valid-unMultigraph.no-id[OF valid-unMultigraph G] \((n1,w,v')\in \text{edges} G\)
by auto
hence odd-v':odd(degree v' \(\text{del-unEdge} n1 w v' G\))
using \((v' \neq n2)\) even-except-two[of v']
valid-graph.E-validD(2)[OF valid-graph G] \((n1, w, v') \in \text{edges} G\)
valid-unMultigraph.del-UnEdge-even'[OF valid-unMultigraph G] \((n1, w, v') \in \text{edges} G\)
\(\exists \text{finite} (\text{edges} G)\)
unfolding odd-nodes-set-def by auto
ultimately have even-except-two':\(\forall n. n \in \text{nodes} \ (\text{del-unEdge} n1 w v' G) \Rightarrow n \neq n2\)
\(\Rightarrow n \neq v' \Rightarrow \text{even}(\text{degree} n (\text{del-unEdge} n1 w v' G))\)
using del-UnEdge-node[of - n1 w v' G] even-except-two valid-unMultigraph.degree-frame[OF valid-unMultigraph G] \((\text{finite} (\text{edges} G)), \text{of} - n1 v' w\)
by force
obtain \(G1 G2\) where
\(G1\)-nodes: \(\text{nodes} G1=\{n. \exists ps. \text{valid-graph.is-path} (\text{del-unEdge} n1 w v' G)\) \(n ps n1\}\)
and \(G1\)-edges: \(\text{edges} G1=\{(n,e,n'). (n,e,n')\in \text{edges} (\text{del-unEdge} n1 w v' G) \land n \in \text{nodes} G1\) \land n'\in \text{nodes} G1\}
and \(G2\)-nodes: \(\text{nodes} G2=\{n. \exists ps. \text{valid-graph.is-path} (\text{del-unEdge} n1 w v' G)\) \(n ps v'\}\)
and \(G2\)-edges: \(\text{edges} G2=\{(n,e,n'). (n,e,n')\in \text{edges} (\text{del-unEdge} n1 w v' G) \land n \in \text{nodes} G2\) \land n'\in \text{nodes} G2\}
and \(G1-G2\)-edges-union: \(\text{edges} G1 \cup \text{edges} G2 = \text{edges} (\text{del-unEdge} n1 w v' G)\)
and \(\text{edges} G1 \cap \text{edges} G2=\{\}\)
and \(G1-G2\)-nodes-union: \(\text{nodes} G1 \cup \text{nodes} G2=\text{nodes} (\text{del-unEdge} n1 w v' G)\)
and \(\text{nodes} G1 \cap \text{nodes} G2=\{\}\)
and valid-unMultigraph G1
and valid-unMultigraph G2
and valid-unMultigraph.connected G1
and valid-unMultigraph.connected $G_2$

using valid-unMultigraph.connectivity-split[OF ⟨valid-unMultigraph $G$; (valid-unMultigraph $G$; not-connected ⟨$n_1, w, v'$⟩∈edges $G$)⟩]

have $n_2$∈nodes $G_2$ using extend-distinct-path
proof −
  have finite (edges (del-unEdge $n_1 w v'$ $G$))
    unfolding del-unEdge-def using (finite (edges $G$)) by auto
  moreover have num-of-odd-nodes (del-unEdge $n_1 w v'$ $G$) = 2
    by (metis ⟨$n_1, w, v'$⟩∈edges $G$; ⟨$v', w, n_1$⟩∈edges $G$) ⟨num-of-odd-nodes $G$ = 2⟩)
    ⟨$v' ≠ n_2$⟩ ⟨valid-graph $G$⟩ del-UnEdge-even-odd delete-edge-sym
    even-except-two
      ⟨finite (edges $G$); finite (nodes $G$); valid-unMultigraph $G$; n1(2) valid-graph.E-validD(2) valid-unMultigraph.no-id⟩
    ultimately have $∃$ ps. valid-unMultigraph.is-trail (del-unEdge $n_1 w v'$ $G$) $n_2$ ps $v'$
      using valid-unMultigraph.path-between-odds[OF valid0.of $n_2 v'$, OF odd-n2 odd-v']
      ⟨$v' ≠ n_2$⟩
      by auto
      hence $∃$ ps. valid-graph.is-path (del-unEdge $n_1 w v'$ $G$) $n_2$ ps $v'$
        by (metis valid0 valid-unMultigraph.is-trail-intro)
      thus $?thesis$ using $G_2$-nodes by auto
qed
have $v'∈nodes G_2$
proof −
  have valid-graph.is-path (del-unEdge $n_1 w v'$ $G$) $v'$ [] $v'$
    by (metis (full-types) ⟨$n_1, w, v'$⟩∈edges $G$; valid-graph $G$; del-UnEdge-node
      valid01 valid-graph.E-validD(2) valid-graph.is-path-simps(1))
    thus $?thesis$ by (metis (lifting) $G_2$-nodes mem-Collect-eq)
qed
have edges-subset:edges (del-unEdge $n_1 w v'$ $G$) ⊂ edges $G$
using ⟨$n_1, w, v'$⟩∈edges $G$; ⟨$v', w, n_1$⟩∈edges $G$;
unfolding del-unEdge-def by auto
hence card (edges $G_1$) < card (edges $G$)
  by (metis $G_1$-$G_2$-edges-union inf-sup-absorb (finite (edges $G$)) less-infI2 psubset-card-mono)
moreover have finite (edges $G_1$)
  by (metis (full-types) $G_1$-$G_2$-edges-union edges-subset finite-Un finite-subset
    ⟨finite (edges $G$); less-imp-le⟩
moreover have finite (nodes $G_1$)
  using $G_1$-$G_2$-nodes-union ⟨finite (nodes $G$)⟩
  unfolding del-unEdge-def by (metis (full-types) finite-Un select-convs(1))
moreover have $n_1$∈nodes $G_1$
proof −
  have valid-graph.is-path (del-unEdge $n_1 w v'$ $G$) $n_1$ [] $n_1$
  qed
by (metis (full-types) del-UnEdge-node n1 (1) valid0' valid-graph.is-path-simps(1))
thus ?thesis by (metis (lifting) G1-nodes mem-Collect-eq)
qed
moreover hence nodes G1 ≠ {} by auto
moreover have num-of-odd-nodes G1 = 0
proof -
  have ∀ n ∈ nodes G1. even (degree n (del-unEdge n1 w v' G))
    using even-except-two' odd-v' odd-n2 ⟨n2 ∈ nodes G2⟩ ⟨nodes G1 ∩ nodes G2 = {}⟩
    ⟨v' ∈ nodes G2⟩
by (metis (full-types) G1-G2-nodes-union Un_iff disjoint_iff_not_equal)
moreover have valid-graph G2
using (valid-unMultigraph G2) valid-unMultigraph-def
by auto
ultimately have ∀ n ∈ nodes G1. even (degree n G1)
by (metis G1-G2-edges-union Int_commute Un_commute ⟨nodes G1 ∩ nodes G2 = {}⟩)
thus ?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 card (edges G2) < card (edges G)
by (metis G1-G2-edges-union finite (edges G) edges-subset inf-sup-absorb
less-infI2
psubset-card mono sup-commute)
moreover have finite (edges G2)
by (metis (full-types) G1-G2-edges-union edges-subset finite-Un ⟨finite (edges G)⟩
  less-le
rev-finite-subset)
moreover have finite (nodes G2)
by (metis (mono-tags) G1-G2-nodes-union del-UnEdge-node le-sup-iff ⟨finite (nodes G)⟩
  rev-finite-subset subsetI)
moreover have nodes G2 ≠ {} using ⟨v' ∈ nodes G2⟩ by auto
moreover have num-of-odd-nodes G2 = 2
proof -
  have ∀ n ∈ nodes G2. n ∉ {n2, v'} ⟹ even (degree n (del-unEdge n1 w v' G))
using even-except-two'
  by (metis (full-types) G1-G2-edges-union Un_iff insert-iff)
moreover have valid-graph G1
using (valid-unMultigraph G1) valid-unMultigraph-def
by auto
ultimately have ∀ n ∈ nodes G2. n ∉ {n2, v'} ⟹ even (degree n 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 ⟨nodes G1 ∩ nodes G2 = {}⟩)
\[ G_1 \cup G_2 = G_2 \]

nodes \( G_2 \rangle = \text{nodes} G_2 \]

\[ G_2 = \{\} \]

nodes \( G_2 \}

nodes \( G_2 \)

G2

\( G_1 \cap \text{nodes} G_1 \cap \text{nodes} G_2 \]

\( v \in \text{nodes} G_2. \text{odd} (\text{degree} v G_2) \)

by (metis (lifting) mem-Collect-eq)

moreover have odd\((\text{degree} n_2 G_2)\)

using sub-graph-degree-frame[\text{of} G_1 G_2 \text{del-unEdge} n_1 w v' G]\n
by (metis (hide-lams, no-types) G1-G2-edges-union \text{\&} \text{nodes} G1 \cap \text{nodes} G2 = \{\})

\langle \text{valid-graph} G1 \rangle \langle n_2 \in \text{nodes} G_2 \rangle \text{inf-assoc inf-bot-right inf-sup-absorb odd-n2 sup-bot-right sup-commute} \]

hence \( n_2 \in \{v \in \text{nodes} G_2. \text{odd} (\text{degree} v G_2)\} \)

by (metis (lifting) \( \langle n_2 \in \text{nodes} G_2 \rangle \) mem-Collect-eq)

moreover have odd\((\text{degree} v' G_2)\)

using sub-graph-degree-frame[\text{of} G_1 G_2 \text{del-unEdge} n_1 w v' G]\n
by (metis G1-G2-edges-union Int-commute Un-commute \text{\&} \text{nodes} G1 \cap \text{nodes} G2 = \{\})

\langle v' \in \text{nodes} G_2. \text{valid-graph} G_1 \rangle \) odd-\( v' \) \]

hence \( v' \in \{v \in \text{nodes} G_2. \text{odd} (\text{degree} v G_2)\} \)

by (metis \( \langle \text{full-types} \rangle \) Collect-conj-eq Collect-mem-eq Int-Collect \( \langle v' \in \text{nodes} G_2 \rangle \) )

ultimately have \( \{v \in \text{nodes} G_2. \text{odd} (\text{degree} v G_2)\} = \{n_2, v'\} \)

using \( \langle \text{finite} (\text{nodes} G_2) \rangle \) by (induct G2, auto)

thus \( ?\text{thesis} \) using \( \langle v' \neq n_2 \rangle \)

unfolding \( \text{num-of-odd-nodes-def odd-nodes-set-def} \) by auto

qed

ultimately obtain \( s \) \( t \) ps2 where

\( s \) \( : s \in \text{nodes} G_2 \text{ odd} (\text{degree} s G_2) \)

and \( t : t \in \text{nodes} G_2 \text{ odd} (\text{degree} t G_2) \)

and \( s \neq t \)

and \( s-ps2-t: \text{valid-unMultigraph.is-Eulerian-trail} G_2 s ps2 t \)

using \( \langle \text{valid-unMultigraph} G_2 \rangle \) \( \langle \text{valid-unMultigraph.connected} G_2 \rangle \) \( \text{less.hyps[of G2]} \)

by auto

moreover have \( \text{valid-graph} G_1 \)

using \( \langle \text{valid-unMultigraph} G_1 \rangle \) \( \text{valid-unMultigraph-def} \) by auto

ultimately have \( \langle s = n_2 \land t = v' \rangle \lor \langle s = v' \land t = n_2 \rangle \)

using odd-n2 odd-\( v' \) even-except-two \( \langle \text{valid-graph} G_1 G_2 \text{del-unEdge} n_1 w v' G\rangle \)

by (metis G1-G2-edges-union G1-G2-nodes-union UnI1 \text{\&} \text{nodes} G1 \cap \text{nodes} G2 = \{\}) \text{inf-commute sup-commute} \]

sup-commute

moreover have \( \text{merge-G1-G2: } \text{\&} \text{nodes} = \text{nodes} G_1 \cup \text{nodes} G_2, \text{edges} = \text{edges} G_1 \cup \text{edges} G_2 \cup \{ (n_1, w, v'), (v', w, n_1) \} = G \)

proof =

have \( \text{edges} \langle \text{del-unEdge} n_1 w v' G \rangle \cup \{ (n_1, w, v'), (v', w, n_1) \} = \text{edges} G \)

using \( \langle n_1, w, v' \rangle \in \text{edges} G \) \( \langle v', w, n_1 \rangle \in \text{edges} G \)

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

moreover have \( \text{nodes} \langle \text{del-unEdge} n_1 w v' G \rangle = \text{nodes} G \)

unfolding \( \text{del-unEdge-def} \) 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 \(\text{edges } G1 \cap \text{edges } G2 = \{\}\) \(\text{nodes } G1 \cap \text{nodes } G2 = \{\}\) \(\text{valid-unMultigraph } G1\) \(\text{valid-unMultigraph } G2\) \(\text{n1}(1)\) \(\text{n2}(1)\) \(\text{n2}(2)\) \(\text{ps1}\) \(\text{s-ps2-t}\) 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 \(\text{edges } G1 \cap \text{edges } G2 = \{\}\) \(\text{nodes } G1 \cap \text{nodes } G2 = \{\}\) \(\text{valid-unMultigraph } G1\) \(\text{valid-unMultigraph } G2\) \(\text{n1}(1)\) \(\text{n1}(2)\) \(\text{n2}(1)\) \(\text{n2}(2)\) \(\text{ps1}\) \(\text{s-ps2-t}\)

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

qed

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

by auto

next

\text{case less}

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

and \( \text{valid-unMultigraph.connected } G \) and \( \text{num-of-odd-nodes } G = \{\}\)

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

proof (rule cases card (\text{nodes } G) = 1)

fix \( v \) assume \( v \in \text{nodes } G \)

assume \( \text{card } (\text{nodes } G) = 1 \)

hence \( \text{nodes } G = \{v\} \)

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

by auto

have \( \text{edges } G = \{\}\)

proof (rule econtr)

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

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

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

by (metis (hide-lams, no-types) append-Nil2 valid-unMultigraph.is-trail-rev

69
valid-unMultigraph.is-trail.simps(1) \langle valid-unMultigraph G \rangle \text{ singletonE}

valid-unMultigraph.is-trail-split valid-unMultigraphsingleton-distinct-path)

\text{thus} \ False \ \text{by} \ \{\text{metis } v \ \langle valid-unMultigraph G \rangle \ \text{ valid-unMultigraph.no-id}\}

\text{qed}

\text{hence} \ valid-unMultigraph.is-Eulerian-circuit \ G \ v \ \emptyset \ v

\text{by} \ \{\text{metis } (\text{nodes } G = \{ v \}) \ \text{ insert-subset } \ \langle valid-unMultigraph G \rangle \ \text{ rem-unPath.simps}(1)\}

\text{subsetI} \ valid-unMultigraph.is-trail.simps(1)

valid-unMultigraph.is-Eulerian-circuit-def

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

\text{thus} \ \exists ps. \ valid-unMultigraph.is-Eulerian-circuit \ G \ v \ ps \ v \ \text{by} \ \text{auto}

\text{next}

\text{fix} \ v \ \text{assume} \ v \in \text{nodes } G

\text{assume} \ \text{card} \ (\text{nodes } G) \neq 1

\text{moreover have} \ \text{card} \ (\text{nodes } G) \neq 0 \ \text{using} \ (\text{nodes } G \neq \{\})

\text{by} \ \{\text{metis } \langle \text{card-eq-0-iff} \ \langle \text{finite } (\text{nodes } G) \rangle \rangle\}

\text{ultimately have} \ \text{card} \ (\text{nodes } G) \geq 2 \ \text{by} \ \text{auto}

\text{then obtain} \ n \ \text{where} \ \text{card} \ (\text{nodes } G) = \text{Suc} \ (\text{Suc } n)

\text{by} \ \{\text{metis } \langle \text{Nat.le-iff-add add-2-eq-Suc} \rangle\}

\text{hence} \ \exists n \in \text{nodes } G. \ n \neq v \ \text{by} \ \{\text{auto dest!, card-eqD}\}

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

\text{proof} =

\text{assume} \ \text{pre}: \exists w \ w'. \ (v, w, v') \in \text{edges } G \ \Longrightarrow \ \text{thesis}

\text{assume} \ \exists n \in \text{nodes } G. \ n \neq v

\text{then obtain} \ ps \ \text{where} \ \text{ps}: \exists v'. \ \text{valid-graph.is-path} \ G \ v \ ps \ v' \wedge \text{ps}\neq\text{Nil}

\text{using} \ valid-unMultigraph-def

\text{by} \ \{\text{metis } \langle \text{full-types} \ v \in \text{nodes } G \rangle \ \langle valid-unMultigraph G \rangle \ \text{valid-graph.is-path.simps}(1)\}

\langle valid-unMultigraph.connected \ G \rangle \ \langle valid-unMultigraph.connected-def \rangle

\text{then obtain} \ v0 \ w \ v' \ \text{where} \ \exists ps'. \ ps = \text{Cons} \ (v0, w, v') \ ps' \ \text{by} \ \{\text{metis } \langle valid-unMultigraph G \rangle \ \text{valid-graph.is-path.simps}(2)\}

\text{hence} \ v0 = v

\text{using} \ valid-unMultigraph-def

\text{by} \ \{\text{metis } \langle valid-unMultigraph G \rangle \ \text{valid-graph.is-path.simps}(2)\}

\text{hence} \ \langle v, w, v' \rangle \in \text{edges } G

\text{using} \ valid-unMultigraph-def

\text{by} \ \{\text{metis } \exists ps'. \ ps = \langle v0, w, v' \rangle \neq ps' \ \langle valid-unMultigraph G \rangle \ \text{ps}

\text{valid-graph.is-path.simps}(2)\}

\text{thus} \ \text{thesis} \ \text{by} \ \{\text{metis} \ \text{pre}\}

\text{qed}

\text{have} \ \text{all-even}: \forall x \in \text{nodes } G. \ \text{even}(\text{degree } x \ G)

\text{using} \ \{\text{finite } (\text{nodes } G) \} \ \{\text{num-of-odd-nodes } G = 0\}

\text{unfolding} \ \text{num-of-odd-nodes-def add-nodes-set-def} \ \text{by} \ \text{auto}

\text{have} \ \text{odd-v}: \ \text{odd} \ (\text{degree } v \ \langle \text{del-anEdge } v \ w \ v' \ G \rangle)

\text{using} \ \{v \in \text{nodes } G \} \ \langle \text{all-even valid-unMultigraph.del-UnEdge-even[OF} \langle valid-unMultigraph G \rangle \ \langle (v, w, v') \in \text{edges } G \rangle \ \langle \text{finite } (\text{edges } G) \rangle\]

70
\begin{align*}
\text{unfolding } & \text{odd-nodes-set-def by auto} \\
\text{have } & \text{odd-}v': \text{ odd } (\text{degree } v' (\text{del-unEdge } v \ w v' G)) \\
\text{using } & \text{valid-unMultigraph.del-UnEdge-even}([-OF \ \text{valid-unMultigraph } G; (v, w, v') \in \text{edges } G]: \\
& \text{all-even } \text{valid-graph.E-validD(2)}[\text{OF} - (v, w, v') \in \text{edges } G]
\end{align*}
\begin{align*}
\text{by } & \text{auto} \\
\text{have } & \text{valid-unMulti-valid-unMultigraph } (\text{del-unEdge } v \ w v' G) \\
\text{by } & \text{(metis del-unEdge-valid valid-unMultigraph G)} \\
\text{moreover have } & \text{valid-graph: valid-graph } (\text{del-unEdge } v \ w v' G) \\
\text{using } & \text{valid-unMultigraph-def del-undirected} \\
\text{by } & \text{(metis valid-unMultigraph G; delete-edge-valid)} \\
\text{moreover have } & \text{fin-E': finite} (\text{edges } (\text{del-unEdge } v \ w v' G)) \\
\text{using } & \text{finite(edges G): unfolding del-UnEdge-def by auto} \\
\text{moreover have } & \text{fin-V': finite} (\text{nodes } (\text{del-unEdge } v \ w v' G)) \\
\text{using } & \text{finite(nodes G): unfolding del-UnEdge-def by auto} \\
\text{moreover have } & \text{less-card:card(edges } (\text{del-unEdge } v \ w v' G))<\text{card(edges G)} \\
\text{unfolding } & \text{del-unEdge-def using } ((v, w, v') \in \text{edges } G) \\
\text{by } & \text{(metis Diff-insert2 card-Diff2-less } \text{finite edges G): valid-unMultigraph G)} \\
\text{select-convs}(2) & \text{valid-unMultigraph.corres} \\
\text{moreover have } & \text{num-of-odd-nodes } (\text{del-unEdge } v \ w v' G) = 2 \\
\text{using } & \text{(valid-unMultigraph G; num-of-odd-nodes } G = 0; v \in \text{nodes } G) \\
\text{all-even } & \text{del-UnEdge-even-even}([-OF \ \text{valid-unMultigraph } G; \text{finite } (\text{edges } G); \text{finite } (\text{nodes } G)] \\
& \langle (v, w, v') \in \text{edges } G] \text{ valid-graph.E-validD(2)}[\text{OF} - (v, w, v') \in \text{edges } G]
\end{align*}
\begin{align*}
\text{unfolding } & \text{valid-unMultigraph-def by auto} \\
\text{moreover have } & \text{valid-unMultigraph.connected } (\text{del-unEdge } v \ w v' G) \\
\text{using } & \text{(finite } (\text{edges } G); \text{finite } (\text{nodes } G); \text{valid-unMultigraph } G) \\
& \text{valid-unMultigraph.connected } G; \\
\text{by } & \text{(metis } (v, w, v') \in \text{edges } G; \text{all-even valid-unMultigraph.del-unEdge-even-connectivity}) \\
\text{moreover have } & \text{nodes(del-unEdge } v \ w v' G) \neq \emptyset \\
\text{by } & \text{(metis } v \in \text{nodes } G; \text{del-UnEdge-node emptyE)} \\
\text{ultimately obtain } & \text{n1 n2 ps where} \\
& \text{n1-n2:} \\
& \text{n1} \in \text{nodes } (\text{del-unEdge } v \ w v' G) \\
& \text{n2} \in \text{nodes } (\text{del-unEdge } v \ w v' G) \\
& \text{odd } \text{(degree n1 } (\text{del-unEdge } v \ w v' G)) \\
& \text{odd } \text{(degree n2 } (\text{del-unEdge } v \ w v' G)) \\
& \text{n1} \neq \text{n2} \\
& \text{and} \\
& \text{ps-eulerian:} \\
& \text{valid-unMultigraph.is-Eulerian-trail } (\text{del-unEdge } v \ w v' G) \text{n1 ps n2} \\
\text{by } & \text{(metis } \text{num-of-odd-nodes del-unEdge } v \ w v' G) = 2; \text{less.hyps(1))}
\end{align*}
have \( n_1 = v \implies n_2 = v' \implies \text{valid-unMultigraph.is-Eulerian-circuit } G \ v \ (ps @ (v',w,v)) \)

using ps eulerian

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

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

using ps eulerian

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

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

by (metis (mono-tags) all-even del-UnEdge-node insert-iff (finite (edges G)) singletonE 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 bool where
non-adj v v' \equiv v \in V \land v' \in V \land v \neq v' \land \neg \text{adjacent } v v'

lemma (in valid-unSimpGraph) no-quad:
assumes \( \forall u. \ v \in V \implies u \in V \implies v \neq u \implies \exists! \ n. \ \text{adjacent } v n \land \text{adjacent } u n \)
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 (adjacent v1 v2) (adjacent v2 v3) (adjacent v3 v4) (adjacent v4 v1) (v2 \neq v4) (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 and f 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 \{x\}∈A card-Diff2-less)
moreover have ∀x∈B. f x∈B ∨ f x≠ x ∨ f (f x) = x
proof
  fix y assume y∈B
  hence y∈A by auto
  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 Diffiff Diff-insert \{f y\} = y \{y\}∈B singleton-iff)
moreover have f y∉ \{x,f x\} by auto
  by (metis B Diffiff \{x\}∈A \{y\}∈B insertCI less.prems(2))
ultimately show False by auto
qed
ultimately show f y∈B \{y\}∉B \{f y\} = y by auto
qed
ultimately have even (card B) by (metis \{finite\} 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\}= by auto
ultimately show \ ?case using B \{finite\} 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;\\ ∀ v u. v\in V \implies u\in V \implies v\neq u \implies \exists! n. adjacent v n ∧ adjacent u n
  and \{finite E\}
  shows \\ ∀ v\in V. even\{degree v G\}
proof
fix \( v \) assume \( v \in V \)

obtain \( f \) where \( f : f = (\lambda n. (\text{SOME} \ v'. \ n \in V \rightarrow n \neq v \rightarrow \text{adjacent} \ n \ v' \land \text{adjacent} \ v \ v')) \) by auto

have \( \land \ n. \ n \in V \rightarrow n \neq v \rightarrow (\exists v'. \ \text{adjacent} \ n \ v' \land \text{adjacent} \ v \ v') \)

proof (rule, rule)
  
  fix \( n \) assume \( n \in V \ n \neq v \)
  hence \( \exists ! v'. \ \text{adjacent} \ n \ v' \land \text{adjacent} \ v \ v' \)
  using friend-assm \( \text{of} \ n \ v \ | v \in V \) unfolding non-adj-def by auto
  thus \( \exists v'. \ \text{adjacent} \ n \ v' \land \text{adjacent} \ v \ v' \) by auto

qed

hence \( f:\exists ; \land \ n. \ (\exists v'. \ n \in V \rightarrow n \neq v \rightarrow \text{adjacent} \ n \ v' \land \text{adjacent} \ v \ v') \) by auto

have \( \forall x \in \{n. \ \text{adjacent} \ v \ n\} . \ f x \in \{n. \ \text{adjacent} \ v \ n\} \land f x \neq x \land f (f x) = x \)

proof (rule, contr)
  
  assume \( f (f x) \neq x \)
  have adjacent \( f (f x) \ (f (f x)) \)
  using someI-ex \( \text{of} \ f, \text{ex}, \text{of} \ f \ x \) by (metis \text{adjacent-V}(\text{of} \ f, \text{ex}, \text{of} \ f \ x) \text{adjacent-no-loop} \text{calculation}(1) f \text{mem-Collect-eq})

moreover have adjacent \( f (f x) \) \( v \)
  using someI-ex \( \text{of} \ f, \text{ex}, \text{of} \ f \ x \) by (metis \text{adjacent-V}(1) \text{adjacent-sym} \text{calculation} f)

moreover have adjacent \( x \ (f x) \)
  using someI-ex \( \text{of} \ f, \text{ex}, \text{of} \ f \ x \) by (metis \text{adjacent-V}(2) \text{adjacent-no-loop} f)

moreover have \( v \neq f x \)
  by (metis \( f x \in \{n. \ \text{adjacent} \ v \ n\} \land f x \neq x \land f (f x) = x \) by auto

qed

moreover have finite \( \{n. \ \text{adjacent} \ v \ n\} \) by (metis \text{adjacent-finite} \text{assms}(2))

ultimately have \( \text{even} \ (\text{card} \ \{n. \ \text{adjacent} \ v \ n\}) \)
  using even-card-set \( \text{of} \ \{n. \ \text{adjacent} \ v \ n\} f \) by auto

thus \( \text{even} (\text{degree} v G) \) by (metis \text{assms}(2) \text{degree-adjacent})

qed
lemma (in valid-unStampGraph) degree-two-windmill:
  assumes friend-assm:\(\forall u.\ v \in V \implies u \in V \implies v \neq u \implies \exists!\ n. \text{ adjacent } v n \land \text{ adjacent } u n\)
  and finite E and card V\(\geq 2\)
  shows \((\exists v \in V. \text{ degree } v G = 2) \iff (\exists \forall n \in V. \ n \neq v \implies \text{ adjacent } v n)\)
proof
  assume \(\exists v \in V. \text{ degree } v G = 2\)
  then obtain v where degree v G=2 by auto
  hence card \{n. adjacent v n\}=2 using degree-adjacent[OF \{finite E\},of v] by auto
  then obtain v1 v2 where v1v2: \{n. adjacent v n\}={v1,v2} and v1\(\neq\)v2
  proof
    obtain v1 S where \{n. adjacent v n\} = insert v1 S and v1 \notin S and card S = 1
    using \{card \{n. adjacent v n\}=2; \text{ card-Suc-eq}\}[of \{n. adjacent v n\} 1] by auto
    then obtain v2 where S=insert v2 \{
      using card-Suc-eq[of S 0] by auto
      hence \{n. adjacent v n\}={v1,v2} and v1\(\neq\)v2
      using \{n. adjacent v n\} = insert v1 S; (v1 \notin S) by auto
      thus \(?thesis using that[of v1 v2] by auto\)
    qed
  have adjacent v1 v2
  proof
    obtain n where adjacent v n adjacent v1 n using friend-assm[of v v1]
    by (metis \{full-types\} adjacent-V(2); adjacent-sym insertII mem-Collect-eq v1v2)
    hence n\in\{n. adjacent v n\} by auto
    moreover have n\(\neq\)v1 by (metis \{adjacent \text{ v n \_ v1}\}; adjacent-no-loop)
    ultimately have n=v2 using v1v2 by auto
    thus \(?thesis by (metis \{adjacent \text{ v1 v n}\}\)
    qed
  have v1v2-adj:\(\forall x \in V. \ x \in \{n. adjacent v1 n\} \cup \{n. adjacent v2 n\}\)
  proof
    fix x assume x\in V
    have x=v \implies x \in \{n. adjacent v1 n\} \cup \{n. adjacent v2 n\}
    by (metis Un_iff adjacent-sym insertII mem-Collect-eq v1v2)
    moreover have x\(\neq\)v \implies x \in \{n. adjacent v1 n\} \cup \{n. adjacent v2 n\}
    proof
      assume x\(\neq\)v
      then obtain y where adjacent v y adjacent x y
      using friend-assm[of v x]
      by (metis Collect-empty-eq \{x \in V\}; adjacent-V(1) all-not-in-conv insertCI v1v2)
      hence y=v1 \lor y=v2 using v1v2 by auto
      thus x \in \{n. adjacent v1 n\} \cup \{n. adjacent v2 n\} using \{adjacent \text{ x y}\}
      by (metis UnII UnII2 adjacent-sym mem-Collect-eq)
    qed
  ultimately show x \in \{n. adjacent v1 n\} \cup \{n. adjacent v2 n\} by auto
qed

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

proof (rule exI[of - v2], rule, rule)

fix \ n assume v1-adj: \{ n. \text{adjacent } v1 \ n \} - \{ v2, v \} = \{ \} and \ n \in V and \ n \neq v

have \ n \in \{ n. \text{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. \text{adjacent } v1 \ n \} by (metis DiffI False \ n \neq v2 empty-iff insert-iff v1-adj)

thus ?thesis by (metis Un-iff \ n \in V v1v2-adj)

qed

thus adjacent v2 n by auto

qed

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

proof (rule exI[of - v1], rule, rule)

fix \ n assume v2-adj: \{ n. \text{adjacent } v2 \ n \} - \{ v1, v \} = \{ \} and \ n \in V and \ n \neq v

have \ n \in \{ n. \text{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. \text{adjacent } v2 \ n \} by (metis DiffI False \ n \neq v1 empty-iff insert-iff v2-adj)

thus ?thesis by (metis Un-iff \ n \in V v1v2-adj)

qed

thus adjacent v1 n by auto

qed

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

\implies False

proof -

assume \{ n. \text{adjacent } v1 \ n \} - \{ v2, v \} \neq \{ \} \ { n. \text{adjacent } v2 \ n \} - \{ v1, v \} \neq \{ \}

then obtain a b where a:a \in \{ n. \text{adjacent } v1 \ n \} - \{ v2, v \}

and b:b \in \{ n. \text{adjacent } v2 \ n \} - \{ v1, v \}

by auto

have a = b \implies False

proof -

assume a = b

have adjacent v1 a using a by auto

moreover have adjacent a v2 using b \ a = b \ adjacent-sym by auto

moreover have a \neq v by (metis DiffD2 \ a = b \ b doubleton-eq-iff insertI1)
moreover have adjacent v2 v
  by (metis (full-types) adjacent-sym inf-sup-aci(5) insertI1 insert-is-Un
mem-Collect-eq v1v2)
moreover have adjacent v v1 by (metis (full-types) insertI1 mem-Collect-eq
v1v2)
ultimately show False using no-quad[OF friend-assm]
  using (v1 ≠ v2) by auto
qed
moreover have a ≠ b ⇒ False
proof -
  assume a ≠ b
moreover have a ∈ V using a by (metis DiffD1 adjacent-V(2) mem-Collect-eq)
moreover have b ∈ V using b by (metis DiffD1 adjacent-V(2) mem-Collect-eq)
ultimately obtain c where adjacent a c adjacent b c
  using friend-assm[of a b] by auto
hence c ∈ \{n. adjacent v1 n\} ∪ \{n. adjacent v2 n\}
  by (metis (full-types) adjacent-V(2) v1v2-adj)
moreover have c ∈ \{n. adjacent v1 n\} ⇒ False
proof -
  assume c ∈ \{n. adjacent v1 n\}
  hence adjacent v1 c by auto
moreover have adjacent c b by (metis (adjacent b c) adjacent-sym)
moreover have adjacent b v2
  by (metis (full-types) Diff-iff adjacent-sym b mem-Collect-eq)
moreover have adjacent v2 v1 by (metis (adjacent v1 v2) adjacent-sym)
moreover have c ≠ v2
proof (rule ccontr)
  assume ¬ c ≠ v2
  hence c = v2 by auto
  hence adjacent v2 a by (metis (adjacent a c) adjacent-sym)
moreover have adjacent v2 v
  by (metis adjacent-sym insert-iff mem-Collect-eq v1v2)
moreover have adjacent v1 v
  using adjacent-sym v1v2 by auto
  moreover have adjacent v1 a by (metis (full-types) Diff-iff a
  mem-Collect-eq)
ultimately have a = v using friend-assm[of v1 v2]
  by (metis (v1 ≠ v2) adjacent-V(1))
  thus False using a by auto
qed
moreover have b ≠ v1 by (metis DiffD2 b insertI1)
ultimately show False using no-quad[OF friend-assm] by auto
qed
moreover have c ∈ \{n. adjacent v2 n\} ⇒ False
proof -
  assume c ∈ \{n. adjacent v2 n\}
  hence adjacent c v2 by (metis adjacent-sym mem-Collect-eq)
moreover have adjacent a c using (adjacent a c) .
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
  hence adjacent v1 b by (metis (adjacent b c) adjacent-sym)
moreover have adjacent v2 v
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 v
by (metis Diff-iff b mem-Collect-eq)
ultimately have b = v using friend-assm[of v1 v2]
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
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]
lemma (in valid-unSimpGraph) regular:
  assumes friend-assm:∀v u. v∈V → u∈V → v≠u → ∃! n. adjacent v n ∧ adjacent u n
  and finite E and finite V and ¬(∃v∈V. degree v G = 2)
  shows ∃k. ∀v∈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 = (λn. (SOME v'. n∈V → n≠u→adjacent n v' ∧ adjacent u v')) by auto
    have ∀n. n∈V → n≠u → (∃v'. adjacent n v' ∧ adjacent u v')
    proof (rule,rule)
      fix n assume n ∈ V n ≠ u
      hence ∃v'. adjacent n v' ∧ adjacent u v' using friend-assm[of n u] ⟨non-adj v w⟩ unfolding non-adj-def by auto
      thus ∃v'. adjacent n v' ∧ adjacent u v' by auto
    qed
  }

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

\begin{itemize}
  \item obtain \( v\text{-adj-def} \) where \( v\text{-adj-def} \) = \( f \) \( v\text{-adj} \) by \( \text{auto} \)
  \item have \( \text{finite } u\text{-adj} \) using \( u\text{-adj} \text{ adjacent-finite}[\text{OF } \text{finite } E] \) by \( \text{auto} \)
  \item have \( \text{finite } v\text{-adj} \) using \( v\text{-adj} \text{ adjacent-finite}[\text{OF } \text{finite } E] \) by \( \text{auto} \)
  \item hence \( \text{finite } v\text{-adj-def} \) using \( v\text{-adj-def} \) adjacent-finite[\text{OF } \text{finite } E] \) by \( \text{auto} \)
\end{itemize}

have \( \text{inj-on } f v\text{-adj} \) unfolding \( \text{inj-on-def} \)

\begin{itemize}
  \item proof (\text{rule ccontr})
  \item assume \( \neg (\forall x \in v\text{-adj}. \forall y \in v\text{-adj}. f x = f y \rightarrow x = y) \)
  \item then obtain \( x y \) where \( x \in v\text{-adj} \) \( y \in v\text{-adj} \) \( f x = f y \) \( x \neq y \) by \( \text{auto} \)
  \item have \( x \in V \) by (\text{metis } (x \in v\text{-adj}) \text{ adjacent-}\text{V}(2) \text{ mem-Collect-eq } v\text{-adj})
  \item moreover have \( x \neq u \) by (\text{metis } (\text{non-adj } v w) (x \in v\text{-adj}) \text{ mem-Collect-eq } v\text{-adj})
\end{itemize}

non-adj-def \( v\text{-adj} \)

ultimately have adjacent \( f(x) \) \( u \) and adjacent \( f(x) \) using \( \text{someI-ex}[\text{OF } f\text{-ex}[\text{of } y]] \) by (\text{metis } \( f \))

\begin{itemize}
  \item hence \( x \neq y \land v \neq f x \land \text{adjacent } v x \land \text{adjacent } x (f x) \land \text{adjacent } (f x) y \land \text{adjacent } y v \land \text{adjacent-sym } (f x \neq v) \)
  \item using (\text{adjacent-def } v\text{-adj}) (x \in v\text{-adj}) (y \in v\text{-adj}) (f x = f y) (x \neq y) \langle \text{adjacent } x (f x) \rangle v\text{-adj}
  \item by \( \text{auto} \)
  \item thus \( \text{False} \) using \( \text{no-quad}[\text{OF } \text{friend-assm}] \) by \( \text{auto} \)
\end{itemize}

qed

then have card \( v\text{-adj} = \text{card } v\text{-adj} u \) by (\text{metis } \text{card-image } v\text{-adj})

\begin{itemize}
  \item moreover have \( v\text{-adj-def} \subseteq u\text{-adj} \)
\end{itemize}

proof

\begin{itemize}
  \item fix \( x \) assume \( x \in v\text{-adj} u \)
  \item then obtain \( y \) where \( y \in v\text{-adj} \)
  \item and \( x = (\text{SOME } v' \cdot y \in V \rightarrow y \neq u \rightarrow \text{adjacent } y v' \land \text{adjacent } u v') \)
  \item using \( \text{f image-def } v\text{-adj-def} \) by \( \text{auto} \)
  \item hence \( y \in V \rightarrow y \neq u \rightarrow \text{adjacent } y x \land \text{adjacent } u x \) using \( \text{someI-ex}[\text{OF } f\text{-ex}[\text{of } y]] \)
  \item by \( \text{auto} \)
  \item moreover have \( y \in V \) by (\text{metis } (y \in v\text{-adj}) \text{ adjacent-}\text{V}(2) \text{ mem-Collect-eq } v\text{-adj})
  \item moreover have \( y \neq u \) by (\text{metis } (\text{non-adj } v w) (y \in v\text{-adj}) \text{ mem-Collect-eq } v\text{-adj})
\end{itemize}

non-adj-def \( v\text{-adj} \)

ultimately have adjacent \( u x \) by \( \text{auto} \)

\begin{itemize}
  \item thus \( x \in u\text{-adj} \) unfolding \( u\text{-adj} \) by \( \text{auto} \)
\end{itemize}

qed

moreover have \( \text{card } v\text{-adj} = \text{degree } v G \) using \( \text{degree-def } v G \) \( \text{degree-adjacent}[\text{OF } \text{finite } E] \), of \( v \) \( v\text{-adj} \) by \( \text{auto} \)

moreover have \( \text{card } u\text{-adj} = \text{degree } u G \) using \( \text{degree-def } v G \) \( \text{degree-adjacent}[\text{OF } \text{finite } E] \), of \( v \) \( v\text{-adj} \) by \( \text{auto} \)
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 v-adj-u)
hence non-adj-degree:∀ u. non-adj v u ⇒ degree v G = degree u G
by (metis adjacent-sym antisym non-adj-def)
have \( \text{card } V = 3 \implies ?\text{thesis} \)
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 \( V S_1 : V = \text{insert } v_1 S_1 \text{ and } v_1 \notin S_1 \text{ and card } S_1 = 2 \)
using card-Suc-eq[of \( V \) 2] (card \( V = 3 \)) by auto
then obtain \( v_2 S_2 \) where \( S_1 S_2 : S_1 = \text{insert } v_2 S_2 \text{ and } v_2 \notin S_2 \) and card \( S_2 = 1 \)
using card-Suc-eq[of \( S_1 \) 1] by auto
then obtain \( v_3 \) where \( S_2 = \{ v_3 \} \)
using card-Suc-eq[of \( S_2 \) 0] by auto
hence \( V = \{ v_1, v_2, v_3 \} \) using \( V S_1 S_2 \) by auto
moreover have \( v_1 \neq v_2 \) \( v_2 \neq v_3 \) \( v_1 \neq v_3 \)
using friend-assm[of \( v_1 v_2 \)] by (metis \( \langle V = \{ v_1, v_2, v_3 \} \rangle \langle v_1 \neq v_2 \rangle \) insertI1
insertI2)
major show \( ?\text{thesis} \) using that by auto
qed
obtain \( n \) where \( \text{adjacent } v_1 n \text{ adjacent } v_2 n \)
using friend-assm[of \( v_1 v_2 \)] by (metis \( \langle V = \{ v_1, v_2, v_3 \} \rangle \langle v_1 \neq v_2 \rangle \) insertI1
insertI2)
major 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)
major obtain \( n' \) where \( \text{adjacent } v_2 n' \text{ adjacent } v_3 n' \)
using friend-assm[of \( v_2 v_3 \)] by (metis \( \langle V = \{ v_1, v_2, v_3 \} \rangle \langle v_2 \neq v_3 \rangle \) insertI1
insertI2)
major 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 \( \text{adjacent } v_1 v_2 \text{ and } \text{adjacent } v_2 v_3 \text{ and } \text{adjacent } v_3 v_1 \)
using adjacent-sym by auto
have \( \text{degree } v_1 G = 2 \)
proof =
have \( v_2 \in \{ n. \text{ adjacent } v_1 n \} \text{ and } v_3 \in \{ n. \text{ adjacent } v_1 n \} \text{ and } v_1 \notin \{ n. \text{ adjacent } v_1 n \} \)
using (adjacent v1 v2, adjacent v3 v1) 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 \( ?\text{thesis} \) using degree-adjacent[OF finite E,of v1 \langle v2 \neq v3 \rangle] by auto
qed
moreover have \( \text{degree } v_2 G = 2 \)
proof =
have \( v_1 \in \{ n. \text{ adjacent } v_2 n \} \text{ and } v_3 \in \{ n. \text{ adjacent } v_2 n \} \text{ and } v_2 \notin \{ n. \text{ adjacent } v_2 n \} \)
have \( v_3 \in \{ n. \text{ adjacent } v_2 n \} \text{ and } v_3 \in \{ n. \text{ adjacent } v_2 n \} \text{ and } v_2 \notin \{ n. \text{ adjacent } v_2 n \} \)
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 = 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:V = insert v1 S1 and v1 ∉ S1 and card S1 = 1
using card-Suc-eq[of V 1] \{ card V = 2 \} by auto
then obtain v2 where S1 = \{ v2 \}
using card-Suc-eq[of S1 0] 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 \( v_1 \) \( \mathcal{G} = 0 \) unfolding degree-def by auto

thus \( \forall v \in V. \) degree \( v \) \( \mathcal{G} = 0 \) using \( \langle V = \{ v_1 \} \rangle \) by auto

cqed

moreover have \( \text{card } V = 0 \implies \text{thesis} \)

proof

assume \( \text{card } V = 0 \)

hence \( V = \{ \} \) using \( \langle \text{finite } V \rangle \) by auto

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

cqed

moreover have \( \text{card } V \geq 4 \implies \neg (\exists v \ u. \ \text{non-adj } v \ u) \implies \neg \text{thesis} \)

proof

assume \( \neg (\exists v \ u. \ \text{non-adj } v \ u) \)

hence \( \text{card } V \geq 4 \) 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 \( V = \text{insert } v_1 \ B_1 \ v_1 \notin B_1 \) card \( B_1 \geq 3 \) finite \( B_1 \) using \( \langle \text{card } V \geq 4 \rangle \) card-le-Suc-iff [OF \( \langle \text{finite } V \rangle , \text{of } 3 \) ] by auto

then obtain \( v_2 \ B_2 \) where \( B_1 = \text{insert } v_2 \ B_2 \ v_2 \notin B_2 \) card \( B_2 \geq 2 \) finite \( B_2 \) using card-le-Suc-iff [of \( B_1 \) 2] by auto

then obtain \( v_3 \ B_3 \) where \( B_2 = \text{insert } v_3 \ B_3 \ v_3 \notin B_3 \) card \( B_3 \geq 1 \) finite \( B_3 \) using card-le-Suc-iff [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 [of \( B_3 \) 0] by auto

have \( v_1 \in V \) by (metis \( \langle V = \text{insert } v_1 \ B_1 \rangle , \text{insert-subset order-refl} \))

moreover have \( v_2 \in V \)

by (metis \( \langle B_1 = \text{insert } v_2 \ B_2 , V = \text{insert } v_1 \ B_1 \rangle , \text{insert-subsetI} \))

moreover have \( v_3 \in V \)

by (metis \( \langle B_1 = \text{insert } v_2 \ B_2 , V = \text{insert } v_1 \ B_1 \rangle , \text{insert-if} \))

moreover have \( v_4 \in V \)

by (metis \( \langle B_1 = \text{insert } v_2 \ B_2 , B_2 = \text{insert } v_3 \ B_3 , V = \text{insert } v_1 \ B_1 \rangle , \text{insert-if} \))

moreover have \( v_1 \neq v_2 \)

by (metis \( \langle \text{full-types} \rangle , \langle B_1 = \text{insert } v_2 \ B_2 , \text{insertI1} \rangle \))

moreover have \( v_1 \neq v_3 \)

by (metis \( \langle B_1 = \text{insert } v_2 \ B_2 , B_2 = \text{insert } v_3 \ B_3 , \text{insertI1} \rangle \))

moreover have \( v_1 \neq v_4 \)

by (metis \( \langle B_1 = \text{insert } v_2 \ B_2 , B_2 = \text{insert } v_3 \ B_3 , B_3 = \text{insert } v_4 \ B_4 , \text{insertI1} \rangle \))

moreover have \( v_2 \neq v_3 \)

by (metis \( \langle \text{full-types} \rangle , \langle B_2 = \text{insert } v_3 \ B_3 , \text{insertI1} \rangle \))

moreover have \( v_2 \neq v_4 \)

by (metis \( \langle B_2 = \text{insert } v_3 \ B_3 , B_3 = \text{insert } v_4 \ B_4 , \text{insertI1} \rangle \))

moreover have \( v_3 \neq v_4 \)
ultimately show \( \text{thesis} \) using that by auto

qed

hence adjacent \( v_1 \) \( v_2 \) using non-non-adj by auto

moreover have adjacent \( v_2 \) \( v_3 \) using non-non-adj by (metis \( v_2 \in V \), \( v_2 \neq v_3 \), \( v_3 \in V \))

moreover have adjacent \( v_3 \) \( v_4 \) using non-non-adj by (metis \( v_3 \in V \), \( v_3 \neq v_4 \), \( v_4 \in V \))

moreover have adjacent \( v_4 \) \( v_5 \) using non-non-adj by (metis \( v_4 \in V \), \( v_4 \neq v_5 \), \( v_5 \in V \))

ultimately show False using no-quad[OF friend-assm]

by (metis \( v_1 \neq v_3 \), \( v_2 \neq v_4 \))

qed

moreover have \( \text{card} \ V \geq 4 \implies (\exists u. \text{non-adj } v \ u) \implies \text{thesis} \)

proof

assume \( \exists v, u. \text{non-adj } v \ u \) card \( V \geq 4 \)

then obtain \( v \ u \) where non-adj \( v \ u \) by auto

then obtain \( w \) where adjacent \( v \ w \) and adjacent \( u \ w \)

and unique:\( \forall n. \text{adjacent } v \ n \land \text{adjacent } u \ n \implies n=w \)

using friend-assm[of \( v \ u \)] unfolding non-adj-def by auto

have \( \forall n \in V. \text{degree } n \ G = \text{degree } v \ G \)

proof

fix \( n \) assume \( n \in V \)

moreover have \( n=v \implies \text{degree } n \ G = \text{degree } v \ G \) by auto

moreover have \( n=u \implies \text{degree } n \ G = \text{degree } v \ G \)

using non-adj-degree \( \langle \text{non-adj } v \ w \rangle \) by auto

moreover have \( n \neq v \implies n \neq u \implies n \neq w \implies \text{degree } n \ G = \text{degree } v \ G \)

proof

assume \( n \neq v \ n \neq u n \neq w \)

have non-adj \( v \ n \implies \text{degree } n \ G = \text{degree } v \ G \) by (metis non-adj-degree)

moreover have non-adj \( u \ n \implies \text{degree } n \ G = \text{degree } v \ G \)

by (metis \( \langle \text{non-adj } v \ w \rangle \ \text{non-adj-degree} \))

moreover have \( \neg \text{non-adj } u \ n \implies \neg \text{non-adj } v \ n \implies \text{degree } n \ G = \text{degree } v \ G \)

by (metis \( \langle n \in V \rangle \ \langle n \neq w \rangle \ \langle \text{non-adj } v \ w \rangle \ \text{non-adj-def} \ \text{unique} \))

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

qed

moreover have \( n=w \implies \text{degree } n \ G = \text{degree } v \ G \)

proof

assume \( n=w \)

moreover have \( \neg (\exists v. \forall n \in V. n \neq v \implies \text{adjacent } v \ n) \)

using \( \langle \text{card } V \geq 4 \rangle \ \text{degree-two-windmill assms(2) assms(4)} \ \text{friend-assm} \)

by auto

ultimately obtain \( w1 \) where \( w1 \in V \ w1 \neq w \) non-adj \( w \ w1 \)

by (metis \( \langle n \in V \rangle \ \langle \text{non-adj-def} \rangle \))

have \( w1=v \implies \text{degree } n \ G = \text{degree } v \ G \)

by (metis \( \langle n = w \rangle \ \langle \text{non-adj } w \ w1 \rangle \ \text{non-adj-degree} \))

moreover have \( w1=u \implies \text{degree } n \ G = \text{degree } v \ G \)

by (metis \( \langle \text{adjacent } u \ w \rangle \ \langle \text{non-adj } w \ w1 \rangle \ \text{adjacent-sym} \ \text{non-adj-def} \))
moreover have \( w_1 \neq u \Rightarrow w_1 \neq v \Rightarrow \text{degree } n \ G = \text{degree } v \ G \)

by (metis \( n = w \) \( \text{non-adj } v \ w \) \( \text{non-adj } w \ w_1 \) \( \text{non-adj-def} \) \( \text{non-adj-degree unique} \))

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

qed

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

qed

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

qed

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

qed

11 Exclusive steps for combinatorial proofs

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

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

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

\[ \text{adj-path } v \ ps \Longrightarrow \text{adj-path } v \ (\text{butlast } ps) \]

by (induct ps arbitrary:v, auto)

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

\[ \text{adj-path } v \ ps \Longrightarrow \text{set } ps \subseteq V \]

by (induct ps arbitrary:v, auto)

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

\[ \text{adj-path } v \ ps \Longrightarrow v \in V \]

by (induct ps arbitrary:v, auto)

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

\[ \text{adj-path } v \ ps \Longrightarrow ps \# [] \Longrightarrow \text{adjacent } (\text{last } ps) \ u \Longrightarrow \text{adj-path } v \ (ps @ [u]) \]

proof (induct ps arbitrary:v)

case Nil

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

next

case \( (\text{Cons } x \ xs) \)

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

qed

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

\[ \text{adj-path } v \ (ps @ [q]) \Longrightarrow ps \neq [] \Longrightarrow \text{adjacent } (\text{last } ps) \ q \]

proof (induct ps arbitrary:v)

case Nil

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

next

case \( (\text{Cons } x \ xs) \)

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

85
qed

lemma card-partition':
  assumes \( \forall v \in A. \text{card} \{ n. R v n \} = k \) \( k > 0 \) finite \( A \)
  \( \forall v1 v2, v1 \neq v2 \rightarrow \{ n. R v1 n \} \cap \{ n. R v2 n \} = \{ \} \)
  shows \text{card} \( \bigcup v \in A. \{ n. R v n \} \) = \( k \ast \text{card} \ A \)
proof
  have \( \bigwedge C. C \in (\lambda x. \{ n. R x n \}) \, \forall A \implies \text{card} \ C = k \)
proof
    fix \( C \) assume \( C \in (\lambda x. \{ n. R x n \}) \, \forall A \)
    shows \text{card} \ C = k \ by \ (\text{metis} \ (\text{mono-tags}) \ C \in (\lambda x. \{ n. R x n \}) \, \forall A \) \text{assms(1)}
imageE
    qed
  moreover have \( \bigwedge C1 C2. C1 \in (\lambda x. \{ n. R x n \}) \, \forall A \implies C2 \in (\lambda x. \{ n. R x n \}) \, \forall A \implies C1 \neq C2 \)
  \( \Rightarrow C1 \cap C2 = \{ \} \)
proof
    fix \( C1 C2 \) assume \( C1 \in (\lambda x. \{ n. R x n \}) \, \forall A \implies C2 \in (\lambda x. \{ n. R x n \}) \, \forall A \implies C1 \neq C2 \)
    obtain \( v1 \) where \( v1 \in A \) \( C1 = \{ n. R v1 n \} \) by \ (\text{metis} \ (\lambda x. \{ n. R x n \}) \, \forall A \) \text{imageE}
    obtain \( v2 \) where \( v2 \in A \) \( C2 = \{ n. R v2 n \} \) by \ (\text{metis} \ (\lambda x. \{ n. R x n \}) \, \forall A \) \text{imageE}
    have \( v1 \neq v2 \) by \ (\text{metis} \ (\lambda x. \{ n. R v1 n \}) \, \forall A \, (\lambda x. \{ n. R v2 n \}) \)
    thus \( C1 \cap C2 = \{ \} \) by \ (\text{metis} \ (\lambda x. \{ n. R v1 n \}) \, \forall A \, (\lambda x. \{ n. R v2 n \}) \) \text{assms(4)}
    qed
  moreover have \( \bigcup (\lambda x. \{ n. R x n \}) \, \forall A = (\bigcup x \in A. \{ n. R x n \}) \) \text{ by auto}
  moreover have \ text{finite} \ (\lambda x. \{ n. R x n \}) \, \forall A \ by \ (\text{metis} \ \text{assms}(3) \ \text{finite-imageI})
  moreover have \text{finite} \ (\bigcup (\lambda x. \{ n. R x n \}) \, \forall A) \ by \ (\text{metis} \ \text{full-types} \ \text{Union-image-eq} \ \text{assms}(1))
  \text{assms(2) assms(3) card-eq-0-iff finite-UN-I less-nat-zero-code}
  moreover have \text{card} \ A = \text{card} \ (\lambda x. \{ n. R x n \}) \, \forall A \)
proof
  have \text{inj-on} \ (\lambda x. \{ n. R x n \}) \, \forall A \ unfolding \text{inj-on-def}
  using \( \forall v1 v2, v1 \neq v2 \rightarrow \{ n. R v1 n \} \cap \{ n. R v2 n \} = \{ \} \)
  by \ (\text{metis} \ \text{assms}(1) \ \text{assms}(2) \ \text{card-empty inf.idem less-le})
  thus \ (\text{?thesis by (metis card-image)})
    qed
ultimately show \ ?thesis using \ (\text{card-partition[of} \ (\lambda x. \{ n. R x n \}) \, \forall A \) \text{ by auto)
qed

lemma \ (\text{in valid-unSimpGraph}) \text{ path-count:}
  assumes \( k \text{-adj} : \forall v. v \in \text{V} \implies \text{card} \{ n. \text{adjacent} v n \} = k \) \( k > 0 \) finite \( \text{V} \)
  \( \forall v1 v2, v1 \neq v2 \rightarrow \{ n. R v1 n \} \cap \{ n. R v2 n \} = \{ \} \)
proof \ (\text{induct} l \text{ rule: nat.induct})
case \text{zero}
  have \( \{ n. \text{length} \ v p s = 0 \} \wedge \text{adj-path} v p s = \{ \} \) using \( v \in \text{V} \) \text{ by auto)
thus \(?case\) by auto

next

case \(\text{Suc } n\)

obtain ext where ext: \(\text{ext}(\lambda ps ps'. ps' \neq [] \land \text{butlast } ps' = ps) \land \text{adj-path } v \text{ ps}\')

by auto

have \(\forall ps \in \{ps. \text{length } ps = n \land \text{adj-path } v \text{ ps}\}. \text{card } \{ps'. \text{ext } ps \text{ ps}'\} = k\)

proof

fix ps assume ps \(\in \{ps. \text{length } ps = n \land \text{adj-path } v \text{ ps}\}\)

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

obtain qs where qs: qs = \{n. if ps = [] then adjacent v n else adjacent \(\text{last } \text{ps}\) n\} by auto

hence card qs = k

proof

\(-\)

have \(\forall xs. xs \in \text{app }' \text{ qs} \implies xs \in \{ps'. \text{ext } ps \text{ ps}'\}\)

proof

rule cases ps = []

\(\text{case True}\)

thus \(?thesis\) using qs k-adj \(\{\text{OF } v \in V\}\) by auto

\(\text{next}\)

\(\text{case False}\)

have \(\text{last } ps \in V\) using adj-path-V by \(\text{metis False } \text{adj-path } v \text{ ps} \text{ last-in-set set-mp}\)

thus \(?thesis\) using k-adj \(\{\text{of last } \text{ps}\}\) False qs by auto

qed

obtain app where app: app = \(\lambda q. p_{s\text{@}}(q)\) by auto

have \(\text{app }' \text{ qs} = \{ps'. \text{ext } ps \text{ ps}'\}\)

proof

\(-\)

have \(\forall xs. xs \in \text{app }' \text{ qs} \implies xs \in \{ps'. \text{ext } ps \text{ ps}'\}\)

proof

\(\text{rule cases ps=[]}\)

\(\text{case True}\)

fix xs assume xs \(\in \text{app }' \text{ qs}\)

then obtain q where q \(\in \text{qs app q=}xs\) by \(\text{metis imageE}\)

hence adjacent v q and xs = ps@\(q\) using qs app True by auto

hence \(\text{adj-path } v \text{ xs}\)

by \(\text{metis True adj-path.simps(1) adj-path.simps(2) adjacent-V(2)}\)

append-Nil)

moreover have \(\text{butlast } xs = ps\) using \(xs=ps@\(q\)) by auto

ultimately show \(\text{ext } ps \text{ xs} \implies \text{ext } ps@\(q\): by auto

next

\(\text{case False}\)

fix xs assume xs \(\in \text{app }' \text{ qs}\)

then obtain q where q \(\in \text{qs app q=}xs\) by \(\text{metis imageE}\)

hence adjacent \(\{\text{last } ps\}\) q using qs app False by auto

hence \(\text{adj-path } v \text{ } \text{ps@\(q\)}\) using \(\langle\text{adj-path } v \text{ ps}\rangle\) False adj-path-app by auto

hence \(\text{adj-path } v \text{ xs}\) by \(\text{metis \(\langle\text{app } q = xs\rangle\)}\)

moreover have \(\text{butlast } xs=ps\) by \(\text{metis \(\langle\text{app } q = xs\rangle\)}\) \(\text{butlast-snoc}\)

ultimately show \(\text{ext } ps \text{ xs}\) by \(\text{metis False } \text{butlast.simps(1) ext}\)

qed

moreover have \(\forall xs. xs \in \{ps'. \text{ext } ps \text{ ps}'\} \implies xs \in \text{app }' \text{ qs}\)

proof

\(\text{cases ps=[]}\)

\(\text{case True}\)

87
hence \( qs = \{ n. \text{ adjacent } v n \} \) using \( qs \) by auto
fix \( xs \) assume \( xs \in \{ ps', \text{ ext } ps ps' \} \)
hence \( xs \neq [] \) and \( \text{ (butlast } xs=ps) \) and \( \text{ adj-path } v xs \) using \( \text{ ext } \) by auto
thus \( xs \in \text{ app ' } qs \)
using True app \( qs = \{ n. \text{ adjacent } v n \} \)
by (metis \( \text{ adj-path.simps(2)} \) append-butlast-last-id append-self-conv2
\[ \text{ image-iff mem-Collect-eq} \]next
case False
fix \( xs \) assume \( xs \in \{ ps', \text{ ext } ps ps' \} \)
hence \( xs \neq [] \) and \( \text{ (butlast } xs=ps) \) and \( \text{ adj-path } v xs \) using \( \text{ ext } \) by auto
then obtain \( q \) where \( xs=ps@[q] \) by (metis \( \text{ append-butlast-last-id} \))
hence \( \text{ adjacent } (last ps) q \) using \( \text{ (adj-path } v xs \) False \( \text{ adj-path-app'} \) by auto
thus \( xs \in \text{ app ' } qs \) using \( qs \)
by (metis \( \text{ (lifting, full-types)} \) False \( \text{ qs } \) using \( ps \) \( \text{ app imageI mem-Collect-eq} \) qed
ultimately show \( ?\text{thesis} \) by auto
qed
moreover have \( \text{ inj-on app } qs \) using \( \text{ app unfolding } \text{ inj-on-def} \) by auto
ultimately show \( \text{ card } \{ ps', \text{ ext } ps ps' \} = k \) by (metis \( \text{ (card } qs = k \) \) card-image)
qed
moreover have \( \forall ps1 ps2. ps1 \neq ps2 \rightarrow \{ \text{ n. ext } ps1 n \} \cap \{ \text{ n. ext } ps2 n \} = \{} \)
using \( \text{ ext } \) by auto
moreover have \( \text{ finite } \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \} \)
using \( \text{ Suc.hyps assms by (auto intro; card-ge-0-finite) } \)
ultimately have \( \text{ card } (\bigcup v \in \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \}. \{ \text{ n. ext } v n \}) = k \ast \text{ card } \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \}
\text{ using } \text{ card-partition’of } \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \} \text{ ext } k ) (k>0) \) by auto
moreover have \( \{ ps. \text{ length } ps = n+1 \wedge \text{ adj-path } v ps \} = (\bigcup ps \in \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \}. \{ ps'. \text{ ext } ps ps' \}) \)
proof –
have \( \bigwedge xs. \text{ xs } \in \{ ps. \text{ length } ps = n + 1 \wedge \text{ adj-path } v ps \} \implies \)
\( \text{ xs } \in (\bigcup ps \in \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \}. \{ ps'. \text{ ext } ps ps' \}) \) proof –
fix \( xs \) assume \( xs \in \{ ps. \text{ length } ps = n + 1 \wedge \text{ adj-path } v ps \} \)
hence \( \text{ length } xs = n + 1 \) and \( \text{ adj-path } v xs \) by auto
hence \( \text{ butlast } xs \in \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \} \)
using \( \text{ adj-path-butlast length-butlast mem-Collect-eq} \) by auto
thus \( xs \in (\bigcup ps \in \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \}. \{ ps'. \text{ ext } ps ps' \}) \)
using \( \text{ (adj-path } v xs \} \text{ length } xs = n + 1 \) UN-iff \( \text{ ext length-greater-0-conv mem-Collect-eq} \) by auto
qed
moreover have \( \bigwedge xs. \text{ xs } \in (\bigcup ps \in \{ ps. \text{ length } ps = n \wedge \text{ adj-path } v ps \}. \{ ps'. \text{ } \}) \)

88
The page contains formal mathematical reasoning with proof obligations and lemmas. The content is a mix of natural language and formal logic, typical of a proof in a formal system like Isabelle/HOL. The text includes definitions, theorems, and proof steps. The page seems to be discussing properties of graphs, possibly related to path-finding algorithms or graph theory, given the context of variables and the logical constructs used.
ultimately show \( \text{hds} = \{ \text{n. adjacent } v \ n \} \) by auto

moreover have inj-on \( \text{hd} \) \( \text{l2-eq-v} \) unfolding inj-on-def

proof (rule+)

fix \( x \ y \) assume \( x \in \text{l2-eq-v} \ y \in \text{l2-eq-v} \) \( \text{hd} \ x = \text{hd} \ y \)

hence \( \text{length} \ x = 2 \) and \( \text{last} \ x = \text{last} \ y \) and \( \text{length} \ y = 2 \)

using \( \text{l2-eq-v} \) by auto

hence \( x!1 = y!1 \)

using \( \text{last-conv-nth} \) [of \( x \)] \( \text{last-conv-nth} \) [of \( y \)] by force

moreover have \( x!0 = y!0 \)

using \( \text{hd-conv-nth} \) [of \( x \)] \( \text{hd-conv-nth} \) [of \( y \)] by (metis \( \text{hd-conv-nth} \) \( \text{length-greater-0-conv} \))

ultimately show \( x = y \) using \( \text{last-conv-nth} \) [of \( x \)] \( \text{last-conv-nth} \) [of \( y \)] by (metis \( \text{One-nat-def} \) \( \text{less-2-cases} \))

moreover have \( x!0 = y!0 \)

using \( \langle \text{hd} \ x = \text{hd} \ y \rangle \) \( \langle \text{length} \ x = 2 \rangle \) \( \langle \text{length} \ y = 2 \rangle \) by (metis \( \text{hd-conv-nth} \) \( \text{length-greater-0-conv} \))

ultimately show \( x = y \) using \( \langle \text{length} \ x = 2 \rangle \) \( \langle \text{length} \ y = 2 \rangle \) by (metis \( \text{One-nat-def} \) \( \text{less-2-cases} \))

qed
by (metis (full-types) \text{adj-path-V} \text{last-in-set} \text{length-0-conv} \text{set-rev-mp} \text{zero-neq-numeral})

thus \( x \in V \land x \neq v \) using \( \langle \text{last ps=x} \rangle \langle \text{last ps\neq v} \rangle \) by auto

qed

moreover have \( \forall x \in \{n. \ n \in V \land n \neq v\} \implies x \in \text{last}' l2-neq-v \)

proof –

fix \( x \) assume \( x:x \in \{n. \ n \in V. \ n \neq v\} \)

then obtain \( y \) where adjacent \( v \ y \) adjacent \( x \ y \)

using friend-assm[of \( v \ x \)] \( \langle v \in V \rangle \) by auto

hence \( \text{adj-path} v \ \langle y.x \rangle \) using adjacent-sym[of \( x \ y \)] by auto

hence \( \langle y.x \rangle \in l2-neq-v \) using \( l2-neq-v \) by auto

thus \( x \in \text{last}' l2-neq-v \) by \( (\text{metis imageI last.simps not-Cons-self2}) \)

qed

ultimately show \( \text{thesis} \) by fast

qed

moreover have \( \text{inj-on} \ \text{last} \ l2-neq-v \) unfolding \( \text{inj-on-def} \)

proof \( \text{(rule,rule,rule)} \)

fix \( x \ y \) assume \( x \in l2-neq-v \ y \in l2-neq-v \ \text{last} \ x = \text{last} \ y \)

hence \( \text{length} \ x \in 2 \) and \( \text{adj-path} v \ x \) and \( \text{last} \ x \neq v \) and \( \text{length} \ y \in 2 \) and

\( \text{adj-path} v \ y \)

and \( \text{last} \ y \neq v \)

using \( l2-neq-v \) by auto

obtain \( x1 \ x2 \ y1 \ y2 \) where \( x:x=x1\ x2 \) and \( y:y=y1\ y2 \)

proof –

\{ fix \( l \) assume \( \text{length} \ l = 2 \)

obtain \( h1 \ t \) where \( l = h1 \# t \) and \( \text{length} \ t = 1 \)

using \( \langle \text{length} \ l = 2 \rangle \) \( \text{Suc-length-conv}[\text{of} \ 1 \ l] \) by auto

then obtain \( h2 \) where \( t = [h2] \)

using \( \text{Suc-length-conv}[\text{of} \ 0 \ l] \) 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 \( \langle \text{length} \ x \in 2 \rangle \langle \text{length} \ y \in 2 \rangle \) by \( \text{metis} \)

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 adjacent \( v \ x1 \) and adjacent \( x2 \ x1 \) and \( x2 \in V \)

using \( \langle \text{adj-path} v \ x \rangle \) \( \text{adjacent-sym} \) by auto

moreover have adjacent \( v \ y1 \) and adjacent \( y2 \ y1 \) and \( y2 \in V \)

using \( \langle \text{adj-path} v \ y \rangle \) \( \text{adjacent-sym} \) by auto

ultimately have \( x1 = y1 \) using friend-assm \( \langle v \in V \rangle \)

by \( (\text{metis} \langle \text{last} \ x = \text{last} \ y \rangle \text{ last-ConsL last-ConsR not-Cons-self2} \ x \ y \)

thus \( x = y \) using \( x \ y \langle \text{last} \ x = \text{last} \ y \rangle \) by auto

qed

ultimately show \( \text{thesis} \) unfolding \( \text{bij-betw-def} \) by auto

qed

hence \( \text{card} l2-neq-v = \text{card} \ \{n. \ n \in V \land n \neq v\} \) by \( (\text{metis bij-betw-same-card}) \)

ultimately have \( \text{card} \ \{n. \ n \in V \land n \neq v\} = k* (k-k) \) 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

91
ultimately show \(?thesis
using \(\text{finite } V\) \(\text{card-Un-disjoint}\) \(\{n \in V. n \neq v\}\) \(\text{finite-Un}
by \text{auto}
qed
ultimately show \(\text{card } V = k\times k - k + 1\) by \text{auto}
qed

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

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

lemma \(\text{in } \text{valid-unSimpGraph}\) \(\text{exist-degree-two}\):
assumes \(\text{friend-assm:} \forall v. u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. \text{ adjacent } v n \wedge \text{adjacent } u n\)
and \(\text{finite } E\) and \(\text{finite } V\) and \(\text{card } V \geq 2\)
shows \(\exists v \in V. \text{ degree } v G = 2\)
proof \((\text{rule ccontr})\)
assume \(\neg (\exists v \in V. \text{ degree } v G = 2)\)
hence \(\forall v. v \in V \implies \text{ degree } v G \neq 2\) by \text{auto}
obtain \(k\) where \(k\text{-adj:} \forall v. v \in V \implies \text{card } \{n. \text{ adjacent } v n\} = k\) using \(\text{regular[OF}\)
by (metis (∃ v∈ V. degree v G = 2)) assms(2) assms(3) degree-adjacent)

have k\geq 4

proof

obtain v1 v2 where v1\in V v2\in V v1\neq v2

using \langle\text{card } V\geq 2\rangle by (metis (∃ v∈ V. degree v G = 2)) assms(2)

degree-two-windmill)

have k\neq 0

proof

assume k=0

obtain v3 where adjacent v1 v3 using friend-assm[OF \langle v1\in V \rangle (v2\in V)

⟨v1\neq v2⟩ by auto

hence \text{card } \{ n. \ adjacent v1 n \} \neq 0 using adjacent-finite[OF \langle \text{finite E} \rangle]

by auto

moreover have \text{card } \{ n. \ adjacent v1 n \} = 0 using k-adj[OF \langle v1\in V \rangle]

by (metis (k = 0) 2)

ultimately show False by simp

qed

moreover have even k using even-degree[OF friend-assm]

by (metis \langle v1\in V \rangle assms(2) degree-adjacent k-adj)

hence k\neq 1 and k\neq 3 by auto

moreover have k\neq 2 using (\\forall v. v \in V \implies \text{degree v G} = 2)\} degree-adjacent

k-adj

by (metis \langle v1\in V \rangle assms(2))

ultimately show ?thesis by auto

qed

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

have T-count:\\\forall l::nat. \text{card } \langle T l \rangle = (k+k\cdot k+1)\cdot k\cdot l using card-partition'

proof

fix l::nat

obtain ext where ext:ext=(\\forall v ps. \text{adj-path} v \ (tl ps) \ \land \text{hd ps}=v \ \land \text{length ps}=l+1) by auto

have \\forall v\in V. \text{card } \{ ps. \ ext v ps \} = k\cdot l

proof

fix v assume v \in V

have \\forall ps. ps\in tl' \\{ ps. \ ext v ps \} \implies ps\in \{ ps. \ length ps=l \\land \text{adj-path} v \ ps \}

proof

fix ps assume ps \in tl' \\{ ps. \ ext v ps \}

then obtain ps' where \text{adj-path} v \ (tl ps') \text{hd ps'}=v \text{length ps'}=l+1

ps=tl ps'

using ext by auto

hence \text{adj-path} v \ ps and \text{length} ps=l by auto

thus ps\in \{ ps. \ length ps=l \\land \text{adj-path} v \ ps \} by auto

qed

moreover have \\forall ps. ps\in \{ ps. \ length ps=l \\land \text{adj-path} v \ ps \} \implies ps\in tl'

\{ ps. \ ext v ps \}

proof


\[\begin{align*}
\text{fix } ps & \text{ assume } ps \in \{ps. \ length \ ps = l \land adj-path \ v \ ps\} \\
\text{hence } length \ ps=\& l \text{ and } adj-path \ v \ ps \text{ by auto} \\
\text{moreover obtain } ps' \text{ where } ps'=v \# ps \text{ by auto} \\
\text{ultimately have } adj-path \ v \ (tl \ ps') \text{ and } hd \ ps'=v \text{ and } length \ ps'=l+1 \\
\text{by } auto \\
\text{thus } ps \in tl \ ' \ {ps. \ ext \ v \ ps} \\
\text{by } (\text{metis } ps' = v \# ps; \ ext \ imageI \ mem-Collect-eq \ list.sel(3)) \\
\text{qed} \\
\text{ultimately have } tl \ ' \ {ps. \ ext \ v \ ps} = \{ps. \ length \ ps=l \land adj-path \ v \ ps\} \\
\text{by fast} \\
\text{moreover have } inj-on \ tl \ \{ps. \ ext \ v \ ps\} \text{ unfolding } inj-on-def \\
\text{proof } (\text{rule,rule,rule}) \\
\text{fix } x \ y \text{ assume } x \in \text{Collect} \ (ext \ v) \ y \in \text{Collect} \ (ext \ v) \ \text{tl } x = tl \ y \\
\text{hence } hd \ x=hd \ y \text{ and } x \not\in \text{ and } y \not\in \text{ using } ext \ by \ auto \\
\text{thus } x=y \text{ using } (\text{tl } x = tl \ y) \text{ by } (\text{metis list.sel(1,3) list.exhaust}) \\
\text{qed} \\
\text{moreover have } card \ \{ps. \ length \ ps=l \land adj-path \ v \ ps\} = k'\ \!l \\
\text{by auto} \\
\text{ultimately show } card \ \{ps. \ ext \ v \ ps\} = k'\ \!l \text{ by } (\text{metis card-image}) \\
\text{qed} \\
\text{moreover have } \forall v1 \ v2. \ v1 \neq v2 \rightarrow \{n. \ ext \ v1 \ n\} \cap \{n. \ ext \ v2 \ n\} = \{\} \\
\text{using } ext \ by \ auto \\
\text{moreover have } (\bigcup v \in V. \ \{n. \ ext \ v \ n\})=T \ l \\
\text{proof } - \\
\text{have } \bigwedge ps. \ ps \in (\bigcup v \in V. \ \{n. \ ext \ v \ n\}) \rightarrow ps \in T \ l \text{ using } T \\
\text{proof } - \\
\text{fix } ps \text{ assume } ps \in (\bigcup v \in V. \ \{n. \ ext \ v \ n\}) \\
\text{then obtain } v \text{ where } v \in V \text{ adj-path } v \ (tl \ ps) \ \text{hd } ps = v \ \text{length } ps = l + 1 \\
\text{using } ext \ by \ auto \\
\text{hence } length \ ps = l+1 \text{ and } adj-path \ (hd \ ps) \ (tl \ ps) \text{ by auto} \\
\text{thus } ps \in T \ l \text{ using } T \text{ by auto} \\
\text{qed} \\
\text{moreover have } \bigwedge ps. \ ps \in T \ l \rightarrow ps \in (\bigcup v \in V. \ \{n. \ ext \ v \ n\}) \\
\text{proof } - \\
\text{fix } ps \text{ assume } ps \in T \ l \\
\text{hence } length \ ps = l+1 \text{ and } adj-path \ (hd \ ps) \ (tl \ ps) \text{ using } T \text{ by auto} \\
\text{moreover then obtain } v \text{ where } v = hd \ ps \ v \in V \\
\text{by } (\text{metis adj-path.simps(1) adj-path.simps(2) adjacent-V(1) list.exhaust}) \\
\text{ultimately show } ps \in (\bigcup v \in V. \ \{n. \ ext \ v \ n\}) \text{ using } ext \ by \ auto \\
\text{qed} \\
\text{ultimately show } ?\text{thesis } by \ auto \\
\text{qed} \\
\text{ultimately have } card \ (T \ l) = card \ V \ast k'\ \!l \\
\text{using } \text{card-partition}[of \ V \ ext \ k'\ \!l] \langle 4 \leq k \rangle \text{ assms(3) mult.commute nat-one-le-power} \\
\end{align*}\]
by auto

moreover have \( \text{card } V = \{k \ast k - k + 1\} \)

using total-v-num[of friend-assm,of k] k-adj degree-adjacent (finite V)

(\(\text{finite } V\)) \(\langle \text{card } V \geq 2 \rangle \langle 4 \leq k \rangle \text{-card-0-iff} \)

by force

ultimately show \( \text{card } (T l) = (k \ast k - k + 1) \ast k \ast l \) by auto

qed

obtain \( C \) where \( C\!::\!(\lambda l::\text{nat}. \{\text{ps. length } ps = l+1 \land \text{adj-path } (hd ps) (tl ps)\}) \)

\(\land \text{adjacent } (last ps) (hd ps)) \) by auto

obtain \( C\!::\!(\lambda l::\text{nat}. \{\text{ps. length } ps = l+1 \land \text{adj-path } (hd ps) (tl ps)\}) \)

\(\land (last ps) = (hd ps)) \) by auto

have \( \bigwedge l::\text{nat}. \text{card } (C (l+1)) = k\ast \text{card } (C\!-\!star l) + \text{card } (T l - C\!-\!star l) \)

proof –

fix \( l::\text{nat} \)

have \( C (l+1) = \{\text{ps. length } ps = l+2 \land \text{adj-path } (hd ps) (tl ps) \land \text{adjacent } (last ps) (hd ps) \) \(\land (\text{last } (\text{butlast } ps)=hd ps) \cup \{\text{ps. length } ps = l+2 \land \text{adj-path } (hd ps) (tl ps) \land \text{adjacent } (last ps) (hd ps) \) \(\land (\text{last } (\text{butlast } ps)=hd ps) \cap \{\text{ps. length } ps = l+2 \land \text{adj-path } (hd ps) (tl ps) \land \text{adjacent } (last ps) (hd ps) \) \(\land (\text{last } (\text{butlast } ps)\neqhd ps) \) \(\langle \text{ps. length } ps = l+2 \land \text{adj-path } (hd ps) (tl ps) \land \text{adjacent } (last ps) (hd ps) \) \(\land (\text{last } (\text{butlast } ps)\neqhd ps) \) \(\rangle \) by auto

moreover have finite (\( C (l+1) \))

proof –

have \( C (l+1) \subseteq T (l+1) \) using \( C \) \( T \) by auto

moreover have \( \{k \ast k - k + 1\} \ast k \ast (l + 1) \neq 0 \) using \( k \geq 4 \) by auto

hence finite (\( T (l+1) \)) using T-count[of l+1] by (metis card-infinite)

ultimately show \( \text{thesis } \) by (metis finite-subset)

qed

ultimately have \( \text{card } (C (l+1)) = \text{card } \{\text{ps. length } ps = l+2 \land \text{adj-path } (hd ps) (tl ps) \) \(\land \text{adjacent } (last ps) (hd ps) \land (\text{last } (\text{butlast } ps)\neqhd ps) \} + \text{card } \{\text{ps. length } ps = l+2 \land \text{adj-path } (hd ps) (tl ps) \land \text{adjacent } (last ps) (hd ps) \land (\text{last } (\text{butlast } ps)\neqhd ps) \}

using card-Un-disjoint[of \( \{\text{ps. length } ps = l + 2 \land \text{adj-path } (hd ps) (tl ps) \) \(\land \text{adjacent } (last ps) (hd ps) \land (\text{last } (\text{butlast } ps) = hd ps) \} \{\text{ps. length } ps = l + 2 \land \text{adj-path } (hd ps) (tl ps) \land \text{adjacent } (last ps) (hd ps) \land (\text{last } (\text{butlast } ps) \neq hd ps) \} \) finite-Un

by auto

moreover have \( \text{card } \{\text{ps. length } ps = l+2 \land \text{adj-path } (hd ps) (tl ps) \) \(\land \text{adjacent } (last ps) (hd ps) \land (\text{last } (\text{butlast } ps) = hd ps) = k \ast \text{card } (C\!-\!star l) \}

95
proof –

obtain ext where ext: ext=\(\lambda ps\ ps'.\ ps'\not=\emptyset\) \& (butlast ps'=ps)
     \& adj-path (hd ps') (tl ps')) by auto
have \(\forall ps\in(C\text{-}\text{star} l).\ card\ \{ps'.\ ext\ ps ps'\} = k\)
proof
  fix ps assume ps\in C\text{-}\text{star} l
  hence length ps = l + 1 and adj-path (hd ps) (tl ps) and last ps =

  using C\text{-}\text{star} by auto
obtain qs where qs:qs={v. adjacent (last ps) v} by auto
obtain app where app:app=(\lambda v. ps@[v]) by auto
have app ' qs = \{ps'. ext ps ps'\}
proof –
  have \(\forall x. x\in\text{app}'qs \implies x\in\{ps'. ext ps ps'\}\)
proof
  fix x assume x \in 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 \langle\text{adj-path (hd ps) (tl ps)}\rangle\text{-}\text{app}

hd-append

  last-tl list.sel(2) tl-append2)
qed

moreover have \(\forall x. x\in\{ps'. ext ps ps'\}\implies x\in\text{app}'qs\)
proof –
  fix x assume x \in \{ps'. ext ps ps'\}
  hence x\not=\emptyset 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 \langle\text{butlast x}=ps\rangle 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 \langle x=x1\#t1\rangle\langle t1=[x2]\rangle by auto
thus adjacent (last ps) (last x)
  using \langle\text{adj-path (hd x) (tl x)}\rangle\langle\text{butlast x}=ps\rangle by auto
next
  case False
  hence tl ps\not=\emptyset
  by (metis \langle length ps = l + 1 \rangle add-0-iff add-cancel-left'
       length-0-cancel length-0 add.commute)
moreover have adj-path (hd x) (tl ps \@ \{last x\})
ultimately have adjacent (last (tl ps)) (last x)
using adj-path-app[of hd x 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 (append-butlast x = ps) (x ≠ []): append-butlast-last-id

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 auto
hence finite (T l) using T-count[of []] by (metis finite-subset)
ultimately show ?thesis by (metis finite-subset)

moreover have C-star l
proof -
  have C-star l ⊆ T l using C-star T by auto
  moreover have (k * k - k + 1) * k ≤ l≠0 using (k≥4) by auto
  hence finite (T l) using T-count[of []] by (metis card-infinite)
  ultimately show ?thesis by (metis finite-subset)
qed

moreover have ∃ ps1 ps2. ps1 ≠ ps2 → {ps'. ext ps ps'} = k by (metis card-image)
proof
  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 -
    have ∃ x. x∈{ps∈ (C-star l). {ps'. ext ps ps'}} ⇒ x∈{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
    qed
    and x ≠ [] and butlast x = ps adj-path (hd x) (tl x)
using $C$-star ext by auto
have $length x = l + 2$
  using $\langle$ butlast $x = ps \rangle$ $\langle length ps = l + 1 \rangle$ length-butlast by auto
moreover have adjacent $(hd x) (tl x)$ by (metis $\langle$adj-path $(hd x)$
moreover have adjacent $(last (butlast x)) (last x)$ using $\langle$adj-path $(hd x)$
proof -
  have $length x \geq 2$ using $\langle length x = l + 2 \rangle$ by auto
  hence adjacent $(last (butlast x)) (last x)$ using $\langle$adj-path $(hd x)$
  by (induct $x$, auto, metis $\langle$adj-path $\rangle$ $\langle$last ps $= hd ps$ $\rangle$ by auto)
moreover have adjacent $(hd x) (last x)$
  using $\langle$butlast $x = ps \rangle$ $\langle length ps = l + 1 \rangle$
  by (cases $x$) auto
  thus $?thesis$ using adjacent-sym by auto
qed
moreover have last $(butlast x) = hd x$
  by (metis $\langle$butlast $x = ps \rangle$ $\langle last ps = hd ps \rangle$ $\langle x \neq [] \rangle$ adjacent-no-loop

ultimately show $length x = l + 2 \land$ adj-path $(hd x) (tl x)$
  $\land$ adjacent $(last x) (hd x) \land last (butlast x) = hd x$
  by auto
qed
moreover have $\forall x. x \in \{ ps. length ps = l + 2 \land$ adj-path $(hd ps) (tl x)$
  $\land$ adjacent $(last x) (hd x) \land last (butlast x) = hd ps \} \implies$
  $x \in (\bigcup ps \in (C$-star $l). \{ ps'. ext ps ps' $\})$
proof -
  fix $x$ assume $x \in \{ ps. length ps = l + 2 \land$ adj-path $(hd ps) (tl ps)$
  $\land$ adjacent $(last ps) (hd ps) \land last (butlast ps) = hd ps \}$
  hence $length x = l + 2$ and adj-path $(hd x) (tl x)$ and adjacent $(last x)$
  by auto
obtain $ps$ where $ps; ps = butlast x$ by auto
have $ps \in C$-star $l$
proof -
  have $length ps = l + 1$ using $ps \langle length x = l + 2 \rangle$ by auto
moreover have $hd ps = hd x$
  using $ps \langle length x = l + 2 \rangle$
  by (metis $\langle$full-types $\rangle$ $\langle$adjacent $(last x) (hd x) \rangle$ adjacent-no-loop
  append-Nil append-butlast-last-id butlast.simps(1) list.sel(1)
hd-append2)
**Theorem**

Let \( l \) be a natural number. Suppose \( l \neq 1 \). Consider a list \( ps \) of length \( l \) and a list \( hd \) of length \( l-1 \) such that \( ps \) is adjacent to \( hd \) and \( hd \) is adjacent to \( ps \). Let \( T \) be a list of length \( l-1 \) and let \( l' \) be the length of \( T \). Define \( l'' \) as the length of \( ps \). If \( l'' \neq l \), then there exists a list \( ps' \) of length \( l+1 \) such that \( ps' \) is adjacent to \( hd \).

**Proof**

1. **Base Case:** When \( l = 2 \), the statement holds trivially.
2. **Inductive Step:** Assume the statement holds for \( l-1 \). We need to show it holds for \( l \).
3. **Case Analysis:**
   - If \( l'' = l \), then \( ps \) is adjacent to \( hd \).
   - If \( l'' < l \), then there exists a list \( ps' \) of length \( l+1 \) such that \( ps' \) is adjacent to \( hd \).
4. **Conclusion:** By induction, the statement holds for all \( l \).
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)

moreover have length ps≠1 using ⟨last ps ≠ hd ps⟩
  by (metis Suc-eq-pla1-left gen-length-code(1) gen-length-def)

hence tl ps≠[] using ⟨length ps=1⟩
  by (metis add-diff-cancel-right length-splice length-tl add.commute)

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 x) 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-snoc)

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∈{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-if

hence adj-path (hd (butlast x)) (tl (butlast x))
using adj-path (hd x) (tl x) by (metis adj-path-butlast buttlast-tl)
moreover have last (butlast x) ≠ hd (butlast x)
using ⟨last (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 t1 t2] by auto
then obtain x3 where t2 = [x3]
using Suc-length-conv[of 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. simp (2) in-set-butlastD last. simp)

last-in-set

length-butlast length-greater-0-conv length-pos-in-set
length-ty 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 $\text{length } x \geq 2$ using $(\text{length } x = l + 2)$ by auto
then obtain $x_1 x_2$ where $x = [x_1, x_2]$
proof
obtain $x_1 t_1$ where $x = x_1 \# t_1$ length $t_1 = 1$
using Suc-length-conv[of $1 \cdot x$] $(\text{length } x = 2)$ by auto
then obtain $x_2$ where $t_1 = [x_2]$
using Suc-length-conv[of $0 \cdot t_1$] by auto
have $x = [x_1, x_2]$ using $(x = x_1 \# t_1) : (t_1 = [x_2])$ by auto
thus $?\text{thesis}$ using that by auto
qed
hence last $(\text{butlast } x) = \text{hd} x$ by auto
thus $?\text{thesis}$ using adj-path-V [OF $\langle \text{adj-path } (hd x) (tl x) \rangle$] by auto

moreover have $\text{hd} (\text{butlast } x) = \text{hd} x$ using $(\text{length } x = l + 2)$ by (metis $\langle \text{adjacent } (last x) (hd x) \rangle$ $\langle \text{adjacent-no-loop append-butlast-last-id} \rangle$
bullast.simps(1) list.sel(1) $\langle \text{hd-append} \rangle$

hence $\text{hd} (\text{butlast } x) \in V$ using adj-path-V $[\langle \text{adj-path } (hd x) (tl x) \rangle] \text{ by auto}$
multiply have $\langle \text{last } (\text{butlast } x) \rangle \neq \text{hd} (\text{butlast } x)$ using $(\text{last } (\text{butlast } x)) \neq \text{hd} (\text{butlast } x) = \text{hd} x$ by auto
ultimately have $\exists! \cdot n. \langle \text{adjacent } (last (\text{butlast } x)) n \rangle$ $\langle \text{adjacent } (hd (\text{butlast } x)) n \rangle$ by $\langle \text{friend-assm} \rangle$ by auto

moreover have $\langle \text{length } x \geq 2 \rangle$ using $(\text{length } x = l + 2)$ by auto
hence $\langle \text{adjacent } (last (\text{butlast } x)) (last x) \rangle$
using $(\text{adj-path } (hd x) (tl x))$
by (induct $x$, auto, metis $\langle \text{full-types} \rangle$ $\langle \text{adj-path-simps(2) append-Nil append-butlast-last-id, metis adj-path-app append-butlast-last-id} \rangle$
multiply have $\langle \text{adjacent } (\text{hd } (\text{butlast } x)) (last x) \rangle$
using $\langle \text{adjacent } (last x) (hd x) \rangle$ $\langle \text{hd } (\text{butlast } x) = \text{hd } x \rangle$ $\langle \text{adjacent-sym} \rangle$ by auto
ultimately have $(\text{SOME } n. \langle \text{adjacent } (last (\text{butlast } x)) n \rangle$ $\langle \text{adjacent } (hd (\text{butlast } x)) n \rangle$ = last $x$
using some1-equality by fast
moreover have $x \in (\text{butlast } x)@[(\text{last } x)]$
by (metis $\langle \text{adjacent } (last (\text{butlast } x)) (last x) \rangle$ $\langle \text{adjacent-no-loop append-butlast-last-id buttlast.simps(1)} \rangle$
ultimately show $?\text{thesis}$ using $\text{app}$ by auto

qed
ultimately show $x \in \text{app}'(T \cdot l - \text{C-star } l)$ by (metis image-iff)

ultimately have $\text{app}'(T \cdot l - \text{C-star } l) = \{ \text{ps. length } ps = l + 2 \land \text{adj-path } (hd ps) (tl ps) \}$ $\langle \text{adjacent } (last ps) (hd ps) \land last (\text{butlast } ps) \neq \text{hd } ps \rangle$ by fast
moreover have $\text{inj-on } \langle \text{app } (T \cdot l - \text{C-star } l) \rangle$ using $\text{app unfolding inj-on-def}$ by auto
ultimately show $?\text{thesis}$ by (metis card-image)
ultimately show $\text{card } (C (l+1)) = k \cdot \text{card } (C\text{-star } l) + \text{card } (T l - C\text{-star } l)$ by auto

**proof**

- fix $l::\text{nat}$
- have $C\text{-star } l \subseteq T l$ using $C\text{-star } T$ by auto
- moreover have $\text{card } (T l) \neq 0$ using $T\text{-count } k \geq 4$ by auto
- hence finite $(T l)$ using $(k \geq 4)$ by (metis card-infinite)
- ultimately have $\text{card } (T l - C\text{-star } l) = \text{card}(T l) - \text{card}(C\text{-star } l)$ by (metis card-Diff-subset rev-finite-subset)
- hence $\text{card } (C (l+1)) = k \cdot \text{card } (C\text{-star } l) + (\text{card } (T l) - \text{card } (C\text{-star } l))$ using $(\forall l::\text{nat}. \text{card } (C (l+1)) = k \cdot \text{card } (C\text{-star } l) + (\text{card } (T l) - \text{card } (C\text{-star } l)))$

**proof**

- also have ...$=k \cdot \text{card } (C\text{-star } l) + \text{card } (T l) - \text{card } (C\text{-star } l)$

**proof**

- have $\text{card } (T l) \geq \text{card } (C\text{-star } l)$
- using $(C\text{-star } l \subseteq T b$ (finite $(T l)$): by (metis card-mono)
- thus $\exists$thesis by auto

**proof**

- also have ...$=k \cdot \text{card } (C\text{-star } l) - \text{card } (C\text{-star } l) + \text{card } (T l)$

**proof**

- have $\text{card } (T l) \geq \text{card } (C\text{-star } l)$
- using $(C\text{-star } l \subseteq T b$ (finite $(T l)$): by (metis card-mono)
- moreover have $k \cdot \text{card } (C\text{-star } l) \geq \text{card } (C\text{-star } l)$ using $(k \geq 4)$ by auto
- ultimately show $\exists$thesis by auto

**proof**

- also have ...$= (k -(1::\text{nat}))*\text{card}(C\text{-star } l)+\text{card}(T l)$ using $(k \geq 4)$
- by (metis monoid-mult-class.mult.left-neutral diff-mult-distrib)
- finally have $\text{card } (C (l+1)) = (k -(1::\text{nat}))*\text{card}(C\text{-star } l)+\text{card}(T l)$
- hence $\text{card } (C (l+1)) \mod (k -(1::\text{nat})) = \text{card}(T l) \mod (k -(1::\text{nat}))$ using $(k \geq 4)$
- by (metis mod-mult-self3 mult.commute)
- also have ...$= (k \cdot k\cdot(l+1))*k \cdot l$ mod $(k -(1::\text{nat}))$ using $T\text{-count } k \cdot l$ by auto
- also have ...$= (k -(1::\text{nat}))*k\cdot(l+1)$ mod $(k -(1::\text{nat}))$ by (metis diff-mult-distrib)
- thus $\exists$thesis by auto

**proof**

- also have ...$= l \cdot k\cdot l$ mod $(k -(1::\text{nat}))$
- by (metis mult-right-eq mod-mult-self1 add.commute mult.commute)
- also have ...$= k\cdot l$ mod $(k -(1::\text{nat}))$ by auto
- also have ...$= (k -(1::\text{nat}))*l$ mod $(k -(1::\text{nat}))$ using $(k \geq 4)$ by auto
- also have ...$= l \cdot l$ mod $(k -(1::\text{nat}))$ by (metis mod-add-self2 add.commute power-mod)
- also have ...$= l$ mod $(k -(1::\text{nat}))$ by auto
also have \( \ldots = 1 \) using \( \langle \geq 4 \rangle \) by auto

finally show \( \operatorname{card} \ (C \ (l+1)) \mod (k-(1::nat)) = 1 \).

qed

obtain \( p::\text{nat} \) where \( \text{prime} \ p \ p \ \text{dvd} \ (k-(1::nat)) \) using \( \langle \geq 4 \rangle \)
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 \( *: \langle \forall x::\text{nat}. \ \operatorname{card} \ (C \ (l+1)) \mod (k-(1::nat))=1 \ \text{mod-mod-cancel[OF \( p \ \text{dvd} \ (k-(1::nat))\]}]\)

\( \langle \text{prime} \ p \rangle \)
by (metis mod-if prime-gt-1-nat)

have \( \operatorname{card} \ (C \ (p-1)) \mod p = 1 \)
proof (cases \( 2 \leq p \))
  case True with \( \langle \text{of} \ p \ - \ 2 \rangle \) show \( \text{?thesis} \)
  by (metis Nat.add-diff-assoc2 add-le-cancel-right diff-diff-left one-add-one p-minus-1)

next
  case False with \( \langle \text{of} \ p \ - \ 2 \rangle \) (\( \langle \text{prime} \ p \rangle \) prime-ge-2-nat show \( \text{?thesis} \)
by blast

qed

moreover have \( \operatorname{card} \ (C \ (p-(1::nat))) \mod p=0 \) using \( C \)
proof (cases)
  have closure1:\( \forall x. \ x \in C \ (p-(1::nat)) \Rightarrow \operatorname{rotate1} \ x \in C \ (p-(1::nat)) \)
  proof (cases)
    fix \( \ x \) assume \( x \in C \ (p-(1::nat)) \)
    hence \( \operatorname{length} \ x = p \) and \( \operatorname{adj-path} \ (hd \ x) \ (tl \ x) \) and \( \operatorname{adjacent} \ (last \ x) \ (hd x) \)
  using \( C \) p-minus-1 by auto

  have \( x\neq[] \) using \( \langle \operatorname{length} \ x=p \rangle \) (\( \langle \text{prime} \ p \rangle \) by auto
  hence \( \operatorname{adjacent} \ (last \ (\operatorname{rotate1} \ x)) \ (hd \ (\operatorname{rotate1} \ x))\)=adjacent \( (hd \ x) \) (hd \( (tl \ x) \))
  by (metis \( \langle \text{adjacent} \ (last \ x) \ (hd \ x) \rangle \) adjacent-no-loop append-Nil

list.sel(1,3)

\( \text{hd-append2 last-snoc list.exhaust \operatorname{rotate1}-hd-tl} \)

also have \( \ldots = \text{True} \) using \( \langle \text{adj-path} \ (hd \ x) \ (tl \ x) \rangle \)
using \( \langle \text{adjacent} \ (last \ x) \ (hd \ x) \rangle \ (x \neq []) \)
by (metis \( \langle \text{adjacent} \ (last \ x) \ (hd \ x) \rangle \) adjacent-no-loop append1-eq-conv

append-Nil
\( \langle \text{append-butlast-last-id list.sel(1,3) list.exhaust} \rangle \)

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

qed

moreover have \( \text{adj-path} \ (hd \ (\operatorname{rotate1} \ x)) \ (tl \ (\operatorname{rotate1} \ x)) \)
proof (cases)
  have \( x\neq[] \) using \( \langle \operatorname{length} \ x=p \rangle \) (\( \langle \text{prime} \ p \rangle \) by auto

104
then obtain y ys where y=hd x ys=tl x by auto
hence adj-path y ys and adjacent (last ys) y and ys≠[]
   by (metis adj-path (hd x) (tl x), metis adjacent (last x) (hd x); y
   = hd x)
   (ys = tl x) adjacent-no-loop list.sel(1,3) last.simps last-tl list.exhaust
   , metis (adjacent (last x) (hd x)); (x ≠ []) (ys = tl x) adjacent-no-loop
list.sel(1,3)
   last-ConsL neq-Nil-cone)
   hence adj-path (hd (rotate1 x)) (tl (rotate1 x))
   = adj-path (hd (ys@[y])) (tl (ys@[y]))
   using x≠[] (y=hd x) (ys=tl x) by (metis rotate1-hd-tl)
   also have ...=adj-path (hd ys) ((tl ys)@[y])
   by (metis (ys ≠ []); hd-append tl-append2)
   also have ...=True
   using adj-path-app[OF adj-path y ys; (ys≠[]); adjacent (last ys) y]
   (ys≠[])
   by (metis adj-path.simps(2) append-Cons list.sel(1,3) list.exhaust)
finally show ?thesis by auto
qed
moreover have length (rotate1 x) = p using length x=p by auto
ultimately show rotate1 x ∈ C (p−(1::nat)) using C-p-minus-1 by auto
qed
have closure:∀ n x. x∈C (p−(1::nat))⇒ rotate n x ∈ C (p−(1::nat))
   proof −
   fix n x assume x∈C (p−(1::nat))
   thus rotate n x ∈ C (p−(1::nat))
   by (induct n,auto,metis One-nat-def closure1)
   qed
obtain r where r:=[(x,y). x∈C (p−(1::nat)) ∧ (∃n<p. rotate n x=y)]
by auto
have (∀ n x. x∈C (p−(1::nat))⇒ p dvd card {y.(∃n<p. rotate n x=y)}
   proof −
   fix x assume x ∈ C (p−(1::nat))
   hence length x=p using C-p-minus-1 by auto
   have {y. (∃ n<p. rotate n x=y)}= (λn. rotate n x)  {0..<p} by auto
   moreover have (∀ n1 n2. n1∈{0..<p} ⇒ n2∈{0..<p} ⇒ n1≠n2 ⇒ rotate n1 x≠rotate n2 x
   by auto
   proof −
   fix n1 n2 assume n1 ∈ {0..<p} n2 ∈ {0..<p} n1 ≠ n2 rotate n1 x
   = rotate n2 x
   { fix n1 n2
     assume n1 ∈ {0..<p} n2 ∈ {0..<p} rotate n1 x = rotate n2 x n1>n2
     obtain s:nat where s*(n1−n2) mod p=1 s>0
     proof −
     have n1−n2>0 and n1−n2<p
     using n1 ∈ {0..<p}; n2 ∈ {0..<p}; n1>n2 by auto
     hence coprime (n1−n2) p using prime p
     by (metis (full-types) gcd.commute nat-dvd-not-less prime-imp-coprime-nat)
     hence ∃ x. ((n1−n2) * x = 1) (mod p) by (metis cong_solve-coprime-nat)
then obtain $s::\text{nat}$ where $s*(n1-n2) \mod p=1$

by (metis (card (C (p-(1::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 $\lnot$thesis using that by auto

qed

have rotate $(s+n1) x=rotate (s*n2) x$

using (rotate n1 x=rotate n2 x)

apply (induct s)

apply (auto simp add: algebra-simps)

by (metis add.commute rotate-rotate)

hence rotate $(s+n1 - s*n2) x = x$

using rotate-diff by auto

hence rotate $(s(n1-n2)) x=x$ by (metis diff-mult-distrib

mult.commute)

hence rotate $1 x = x$ using $(s(n1-n2) \mod p=1) \langle \text{length } x=p \rangle$

by (metis rotate-conv-mod)

hence rotate $1 x=x$ by auto

have $(hd x=hd (tl x))$ using $(prime p) \langle \text{length } x=p \rangle$

proof –

have $(\text{length } x\geq 2)$ using $(prime p) \langle \text{length } x=p \rangle$ by auto

hence $(\text{length } (tl x)\geq 1)$ by force

hence $x\not\in\{\emptyset\}$ by auto+

hence $(hd x)\#(hd (tl x))\#(tl (tl x))$ using $\text{hd-Cons-tl}$ by auto

hence $(hd (tl x))\#(tl (tl x))\#[hd x]=(hd x)\#(hd (tl x))\#(tl (tl x))$

using $(\text{rotate}1 x = x)$ by (metis $\text{Cons-eq-append1 rotate1.simps(2)}$)

thus $?thesis$ by auto

qed

moreover have $(hd x\not\in hd (tl x))$

proof –

have $(\text{adj-path} (hd x) (tl x))$ using $(x \in C (p-(1::nat)))$ by auto

moreover have $(\text{length } x\geq 2)$ using $(prime p) \langle \text{length } x=p \rangle$ by auto

hence $(\text{length } (tl x)\geq 1)$ by force

hence $(tl x\not\in\{\emptyset\})$ by force

ultimately have $(\text{adjacent} (hd x) (hd (tl x)))$

by (metis $\text{adj-path.simps(2) list.sel(1) list.exhaust}$)

thus $?thesis$ by (metis $\text{adjacent-no-loop}$)

qed

ultimately have $(False)$ by auto

thus $(False)$

by $(metis (n1 \in \{0..<p\}) \langle n1 \neq n2 \rangle \langle n2 \in \{0..<p\} \rangle \langle \text{rotate } n1 x = rotate n2 x \rangle)$

less-linear)

qed

hence $\text{inj-on} (\lambda n. \text{rotate } n x) \{0..<p\}$ unfolding $\text{inj-on-def}$ by fast

ultimately have $\text{card} \{ y. (\exists n<p. \text{rotate } n x=y) \}=\text{card} \{0..<p\}$ by (metis card-image)
hence \( \text{card} \{ y. (\exists n<p. \text{rotate} n \ x=y)\} = p \) by auto

thus \( p \ \text{ded} \ \text{card} \{ y. (\exists n<p. \text{rotate} n \ x=y)\} \) by auto

\textbf{qed}

hence \( \forall X \in C \ (p-(1::\text{nat})) \) // r. \( p \ \text{ded} \ X \) unfolding \text{quotient-def}

\textbf{Image-def} by auto

moreover have \( \text{refl-on} \ (C \ (p - 1)) \ r \)

\textbf{proof} –

\begin{align*}
& \text{have } r \subseteq C \ (p - 1) \times C \ (p - 1) \\
& \text{proof} \\
& \text{fix } x \ \text{assume } x \in r \\
& \text{hence } \text{fst} \ x \in C \ (p - 1) \text{ and } \exists n. \ \text{snd} \ x = \text{rotate} \ n \ (\text{fst} \ x) \ \text{using } r \text{ by auto}
\end{align*}

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 \text{closure} by auto

moreover have \( x = (\text{fst} \ x, \text{snd} \ x) \) using \( \langle x \in r \rangle \) by auto

ultimately show \( x \in C \ (p - 1) \times C \ (p - 1) \) using \( \langle \text{fst} \ x \in C \ (p - 1) \rangle \)

by (metis \text{SigmaI})

\textbf{qed}

moreover have \( \forall x \in C \ (p - 1). \ (x, x) \in r \)

\textbf{proof} –

\begin{align*}
& \text{fix } x \ \text{assume } x \in C \ (p - 1) \\
& \text{hence } \text{rotate} \ 0 \ x \in C \ (p - 1) \text{ using } \text{closure} \text{ by auto} \\
& \text{moreover have } \theta < p \ \text{using } \theta - \text{prime} p \text{ by auto} \\
& \text{ultimately have } (x, \text{rotate} \ 0 \ x) \in r \text{ using } (x \in C \ (p - 1)) \text{ by auto} \\
& \text{moreover have } \text{rotate} \ 0 \ x = x \text{ by auto} \\
& \text{ultimately show } (x, x) \in r \text{ by auto} \\
& \textbf{qed} \\
& \text{ultimately show } ?\text{thesis} \text{ using } \text{refl-on-def} \text{ by auto} \\
& \textbf{qed}
\end{align*}

moreover have \( \text{sym} \ r \) unfolding \text{sym-def}

\textbf{proof} (\text{rule}, \text{rule}, \text{rule})

\begin{align*}
& \text{fix } x \ y \ \text{assume } (x, y) \in r \\
& \text{hence } x \in C \ (p - 1) \text{ using } r \text{ by auto} \\
& \text{hence } \text{length} \ x = p \text{ using } C \ p - \text{minus-1} \text{ by auto} \\
& \text{obtain } n \ \text{where } n < p \ \text{rotate} \ n \ x = y \text{ using } ((x, y) \in r) \ r \text{ by auto} \\
& \text{hence } y \in C \ (p - 1) \text{ using } \text{closure}[\text{OF } (x \in C \ (p - 1)) \text{]} \text{ by auto} \\
& \text{have } n = 0 \implies (y, x) \in r \\
& \text{proof} – \\
& \text{assume } n = 0 \\
& \text{hence } x = y \text{ using } (\text{rotate} \ n \ x = y) \text{ by auto} \\
& \text{thus } (y, x) \in r \text{ using } (\text{refl-on} \ (C \ (p - 1)) \ r) \ (y \in C \ (p - 1)) \text{ refl-on-def} \\
& \textbf{by fast}
\end{align*}

\textbf{qed}

moreover have \( n \neq 0 \implies (y, x) \in r \)

\textbf{proof} –

\begin{align*}
& \text{assume } n \neq 0 \\
& \text{have } \text{rotate} \ (p - n) \ y = x \\
& \text{proof} –
\end{align*}
have rotate \((p-n)\) \(y = \text{rotate} \ (p-n) \ (\text{rotate} \ n \ x)\) using \(\langle \text{rotate} \ n \ x = y \rangle\) by auto
also have rotate \((p-n)\) \((\text{rotate} \ n \ x) = \text{rotate} \ ((p-n)+n) \ x)\) using \(\text{rotate}-\text{rotate}\) 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 \(\ (y, x) \in r\) using \(\langle y \in C \ (p-1) \rangle\) by auto

qed

ultimately show \(\ (y, x) \in r\) by auto

qed

moreover have \(\text{trans} \ r\) unfolding \(\text{trans-def}\)
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 \(x \in C \ (p-1)\) using \(r\) by auto
hence \(\text{length} \ x = p\) using \(\text{C-p-minus-1}\) by auto
obtain \(n, n2\) where \(n1 < n\) \(n2 < p\) \(y = \text{rotate} \ n1 \ x = \text{rotate} \ n2 \ y\)
using \(r\) \(\langle (x, y) \in r \rangle \ (y, z) \in r\) by auto
hence \(z = \text{rotate} \ ((n2+n1) \ x)\) by (metis \(\text{rotate}-\text{rotate}\))

moreover have \(\text{finite} \ (C \ (p-1))\)
by (metis \(\text{card} \ (C \ (p-1))\) \(\text{mod} \ p = 1\) \(\text{card-eq-0-iff} \ \text{mod-0} \ \text{zero-neq-one}\))
ultimately have \(p \ \text{dvd} \ \text{card} \ (C \ (p-(1::nat)))\) using \(\text{equiv-imp-dvd-card}\)
equiv-def by fast
thus \(\text{card} \ (C \ (p-(1::nat)))\) \(\text{mod} \ p = 0\) by (metis \(\text{dvd-eq-mod-0}\))

qed

ultimately show \(\text{False}\) by auto

qed

theorem \(\langle \text{in valid-unSimpGraph}\rangle\) friendship-thm:
assumes friend-assm:\(\forall \ u, \ v \in V \ \Rightarrow u \in V \ \Rightarrow v \neq u \ \Rightarrow \exists ! \ n. \ \text{adjacent} \ v \ n\)
adjacent \(u \ n\)
and \(\text{finite} \ V\)
shows \(\exists v. \forall n \in V. \ n \neq v \ \Rightarrow \text{adjacent} \ v \ n\)

proof —
have \(\text{card} \ V = 0\) \(\Rightarrow \ ?\text{thesis}\)
using \(\text{finite} \ V\)
by (metis \(\text{all-not-in-conv} \ \text{card-seteq} \ \text{empty-subsetI} \ \text{le0}\))
moreover have \(\text{card} \ V = 1\) \(\Rightarrow \ ?\text{thesis}\)
proof —
assume \(\text{card} \ V = 1\)

108
then obtain \( v \) where \( V=\{v\} \)

using \( \text{card-eq-SucD}[\text{of } V \emptyset] \) by \( \text{auto} \)

hence \( \forall n \in V. \ n = v \) by \( \text{auto} \)

thus \( \exists v. \ \forall n \in V. \ n \neq v \implies \text{adjacent } v \ n \) by \( \text{auto} \)

\( \text{qed} \)

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 \( \text{exist-degree-two}[\text{OF friend-assm}][\text{finite } V] \) by \( \text{auto} \)

thus \( \text{thesis} \)

using \( \text{degree-two-windmill}[\text{OF friend-assm}][\text{card } V \geq 2][\text{finite } V] \) by \( \text{auto} \)

\( \text{qed} \)

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

\( \text{qed} \)

end

References


