

The Königsberg Bridge Problem and the Friendship Theorem

Wenda Li

December 14, 2021

Abstract

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

Contents

1 Undirected Multigraph and undirected trails	2
2 Degrees and related properties	2
3 Connectivity	30
4 Adjacent nodes	45
5 Undirected simple graph	46
6 Definition of Eulerian trails and circuits	49
7 Necessary conditions for Eulerian trails and circuits	49
8 Specific case of the Königsberg Bridge Problem	51
9 Sufficient conditions for Eulerian trails and circuits	53
10 Common steps	72

theory *MoreGraph* **imports** *Complex-Main Dijkstra-Shortest-Path.Graph*
begin

1 Undirected Multigraph and undirected trails

locale *valid-unMultigraph=valid-graph* *G* **for** $G :: ('v, 'w)$ *graph+*
assumes *corres[simp]*: $(v, w, u') \in \text{edges } G \longleftrightarrow (u', w, v) \in \text{edges } G$
and *no-id[simp]*: $(v, w, v) \notin \text{edges } G$

fun (**in** *valid-unMultigraph*) *is-trail* :: $'v \Rightarrow ('v, 'w)$ *path* $\Rightarrow 'v \Rightarrow \text{bool}$ **where**
is-trail $v \ [] \ v' \longleftrightarrow v=v' \wedge v' \in V \ |$
is-trail $v \ ((v1, w, v2) \# ps) \ v' \longleftrightarrow v=v1 \wedge (v1, w, v2) \in E \wedge$
 $(v1, w, v2) \notin \text{set } ps \wedge (v2, w, v1) \notin \text{set } ps \wedge \text{is-trail } v2 \ ps \ v'$

2 Degrees and related properties

definition *degree* :: $'v \Rightarrow ('v, 'w)$ *graph* $\Rightarrow \text{nat}$ **where**
degree $v \ g \equiv \text{card}(\{e. e \in \text{edges } g \wedge \text{fst } e = v\})$

definition *odd-nodes-set* :: $('v, 'w)$ *graph* $\Rightarrow 'v$ *set* **where**
odd-nodes-set $g \equiv \{v. v \in \text{nodes } g \wedge \text{odd}(\text{degree } v \ g)\}$

definition *num-of-odd-nodes* :: $('v, 'w)$ *graph* $\Rightarrow \text{nat}$ **where**
num-of-odd-nodes $g \equiv \text{card}(\text{odd-nodes-set } g)$

definition *num-of-even-nodes* :: $('v, 'w)$ *graph* $\Rightarrow \text{nat}$ **where**
num-of-even-nodes $g \equiv \text{card}(\{v. v \in \text{nodes } g \wedge \text{even}(\text{degree } v \ g)\})$

definition *del-unEdge* **where** *del-unEdge* $v \ e \ v' \ g \equiv ()$
 $\text{nodes} = \text{nodes } g, \text{edges} = \text{edges } g - \{(v, e, v'), (v', e, v)\} \ ()$

definition *rev-path* :: $('v, 'w)$ *path* $\Rightarrow ('v, 'w)$ *path* **where**
rev-path $ps \equiv \text{map } (\lambda(a, b, c). (c, b, a)) (\text{rev } ps)$

fun *rem-unPath*:: $('v, 'w)$ *path* $\Rightarrow ('v, 'w)$ *graph* $\Rightarrow ('v, 'w)$ *graph* **where**
rem-unPath $[] \ g = g$
rem-unPath $((v, w, v') \# ps) \ g =$
rem-unPath $ps \ (\text{del-unEdge } v \ w \ v' \ g)$

lemma *del-undirected*: *del-unEdge* $v \ e \ v' \ g = \text{delete-edge } v' \ e \ v \ (\text{delete-edge } v \ e \ v'$
 $g)$

unfolding *del-unEdge-def delete-edge-def* **by** *auto*

lemma *delete-edge-sym*: *del-unEdge* $v \ e \ v' \ g = \text{del-unEdge } v' \ e \ v \ g$

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*: $\{x \in A - B. P x\} = \{x \in A. P x\} - \{x \in B. P x\}$ **by** *blast*

lemma *set-compre-subset*: $B \subseteq A \implies \{x \in B. P x\} \subseteq \{x \in A. P x\}$ **by** *blast*

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

proof –

assume *assms: finite (edges g) (v, e, v') \in edges g (v', e, v) \in edges g*

have $degree v (del-unEdge v e v' g) + 1$

$= card (\{ea \in edges g - \{(v, e, v'), (v', e, v)\}. fst ea = v\}) + 1$

unfolding *del-unEdge-def degree-def* **by** *simp*

also have $... = card (\{ea \in edges g. fst ea = v\} - \{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\}) + 1$

by (*metis set-compre-diff*)

also have $... = card (\{ea \in edges g. fst ea = v\}) - card (\{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\}) + 1$

proof –

have $\{(v, e, v'), (v', e, v)\} \subseteq edges g$ **using** $\langle (v, e, v') \in edges g \rangle \langle (v', e, v) \in edges g \rangle$

by *auto*

hence $\{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\} \subseteq \{ea \in edges g. fst ea = v\}$

by *auto*

moreover have $finite \{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\}$ **by** *auto*

ultimately have $card (\{ea \in edges g. fst ea = v\} - \{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\}) = card \{ea \in edges g. fst ea = v\} - card \{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\}$

$= card \{ea \in edges g. fst ea = v\} - card \{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\}$

$= card \{ea \in edges g. fst ea = v\} - card \{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\}$

using *card-Diff-subset* **by** *blast*

thus *?thesis* **by** *auto*

qed

also have $... = card (\{ea \in edges g. fst ea = v\})$

proof –

have $\{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\} = \{(v, e, v')\}$ **by** *auto*

hence $card \{ea \in \{(v, e, v'), (v', e, v)\}. fst ea = v\} = 1$ **by** *auto*

moreover have $card \{ea \in edges g. fst ea = v\} \neq 0$

by (*metis* (*lifting*, *mono-tags*) *Collect-empty-eq* *assms(1)* *assms(2)*
card-eq-0-iff *fst-conv* *mem-Collect-eq* *rev-finite-subset* *subsetI*)
ultimately show *?thesis* **by** *arith*
qed
finally have $\text{degree } v \text{ (del-unEdge } v \ e \ v' \ g) + 1 = \text{card } (\{ea \in \text{edges } g. \text{fst } ea = v\})$.
thus *?thesis* **unfolding** *degree-def* .
qed

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

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

using *del-edge-undirected-degree-plus* **by** (*metis* *add-diff-cancel-left'* *add.commute*)

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

lemma *del-unEdge-com*: $\text{del-unEdge } v \ w \ v' \text{ (del-unEdge } n \ e \ n' \ g)$
 $= \text{del-unEdge } n \ e \ n' \text{ (del-unEdge } v \ w \ v' \ g)$
unfolding *del-unEdge-def* **by** *auto*

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

proof (*induct* *ps* *arbitrary*: *g*)

case *Nil*

thus *?case* **by** (*metis* *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]*:

$\text{valid-unMultigraph } g \implies \text{valid-unMultigraph } (\text{rem-unPath } ps \ g)$

proof (*induct* *ps*)

case *Nil*

thus *?case* **by** *simp*

next

case (*Cons* *x* *xs*)

thus *?case*

proof –

```

have valid-unMultigraph (rem-unPath (x # xs) g) = valid-unMultigraph
  (del-unEdge (fst x) (fst (snd x)) (snd (snd x)) (rem-unPath xs g))
  using rem-unPath-com by (metis prod.collapse rem-unPath.simps(2))
also have ...=valid-unMultigraph (rem-unPath xs g)
  by (metis Cons.hyps Cons.premis del-unEdge-valid)
also have ...=True
  using Cons by auto
finally have ?case=True .
thus ?case by simp
qed
qed

```

```

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

```

```

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

```

```

lemma del-UnEdge-node[simp]: v∈nodes (del-unEdge u e u' G) ⟷ v∈nodes G
  by (metis del-unEdge-def select-convs(1))

```

```

lemma [intro!]: finite (edges G) ⟹ finite (edges (del-unEdge u e u' G))

```

by (metis del-unEdge-def finite-Diff select-convs(2))

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

lemma [intro!]: $\text{finite (edges } G) \implies \text{finite (edges (rem-unPath } ps \ G))}$

proof (induct ps arbitrary:G)

case Nil

thus ?case by simp

next

case (Cons x xs)

hence $\text{finite (edges (rem-unPath (x \# \ xs) \ G))} = \text{finite (edges (del-unEdge (fst x) (fst (snd x)) (snd (snd x)) (rem-unPath xs \ G)))}$

by (metis rem-unPath.simps(2) rem-unPath-com surjective-pairing)

also have $\dots = \text{finite (edges (rem-unPath xs \ G))}$

using del-unEdge-def

by (metis finite.emptyI finite-Diff2 finite-Diff-insert select-convs(2))

also have $\dots = \text{True}$ using Cons by auto

finally have ?case = True .

thus ?case by simp

qed

lemma del-UnEdge-frame[intro]:

$x \in \text{edges } g \implies x \neq (v, e, v') \implies x \neq (v', e, v) \implies x \in \text{edges (del-unEdge } v \ e \ v' \ g)$

unfolding del-unEdge-def by auto

lemma [intro!]: $\text{finite (nodes } G) \implies \text{finite (odd-nodes-set } G)$

by (metis (lifting) mem-Collect-eq odd-nodes-set-def rev-finite-subset subsetI)

lemma [simp]: $\text{nodes (del-unEdge } u \ e \ u' \ G) = \text{nodes } G$

by (metis del-unEdge-def select-convs(1))

lemma [simp]: $\text{nodes (rem-unPath } ps \ G) = \text{nodes } G$

proof (induct ps)

case Nil

show ?case by simp

next

case (Cons x xs)

have $\text{nodes (rem-unPath (x \# \ xs) \ G)} = \text{nodes (del-unEdge (fst x) (fst (snd x)) (snd (snd x)) (rem-unPath xs \ G))}$

by (metis rem-unPath.simps(2) rem-unPath-com surjective-pairing)

also have $\dots = \text{nodes (rem-unPath xs \ G)}$ by auto

also have $\dots = \text{nodes } G$ using Cons by auto

finally show ?case .

qed

lemma [intro!]: $\text{finite (nodes } G) \implies \text{finite (nodes (rem-unPath } ps \ G))}$ by auto

lemma in-set-rev-path[simp]: $(v', w, v) \in \text{set (rev-path } ps) \longleftrightarrow (v, w, v') \in \text{set } ps$

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

lemma *rem-unPath-edges*:

$$\text{edges}(\text{rem-unPath } ps \ G) = \text{edges } G - (\text{set } ps \cup \text{set } (\text{rev-path } ps))$$

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

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 $set (rev-path ps) \cup set (rev-path (rev-path ps)) = set ps \cup set (rev-path ps)$
by *auto*
thus *?thesis* **by** (*metis rem-unPath-edges*)
qed
ultimately show *?thesis* **by** *auto*
qed

lemma *distinct-rev-path*[simp]: $distinct (rev-path ps) \longleftrightarrow distinct ps$
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 $... = (distinct (rev-path xs) \wedge (x3, x2, x1) \notin set (rev-path xs))$
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*: $is-path\ v' (rev-path\ ps)\ v \longleftrightarrow is-path\ v\ ps\ v'$
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 $... = (is-path\ v' (rev-path xs)\ x3 \wedge (x3, x2, x1) \in E \wedge is-path\ x1 []\ v)$ **by** *auto*
also have $... = (is-path\ x3\ xs\ v' \wedge (x3, x2, x1) \in E \wedge is-path\ x1 []\ v)$ **using** *Cons.hyps*
by *auto*
also have $... = is-path\ v (x \# xs)\ v'$

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

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

lemma (in valid-unMultigraph) is-trail-path:
 $\text{is-trail } v \text{ ps } v' \longleftrightarrow \text{is-path } v \text{ ps } v' \wedge \text{distinct ps} \wedge (\text{set ps} \cap \text{set (rev-path ps)}) = \{\}$

proof (induct ps arbitrary:v)

case Nil

show ?case by auto

next

case (Cons x xs)

obtain x1 x2 x3 where x: x=(x1,x2,x3) by (metis prod-cases3)

hence is-trail v (x#xs) v' = (v=x1 \wedge (x1,x2,x3) \in E \wedge

(x1,x2,x3) \notin set xs \wedge (x3,x2,x1) \notin set xs \wedge is-trail x3 xs v')

by (metis is-trail.simps(2))

also have ...=(v=x1 \wedge (x1,x2,x3) \in E \wedge (x1,x2,x3) \notin set xs \wedge (x3,x2,x1) \notin set xs \wedge is-path x3 xs v')

\wedge distinct xs \wedge (set xs \cap set (rev-path xs)={\})

using Cons.hyps by auto

also have ...=(is-path v (x#xs) v' \wedge (x1,x2,x3) \neq (x3,x2,x1) \wedge (x1,x2,x3) \notin set xs

\wedge (x3,x2,x1) \notin set xs \wedge distinct xs \wedge (set xs \cap set (rev-path xs)={\}))

by (metis append-Nil is-path.simps(1) is-path.simps(2) is-path-split' no-id x)

also have ...=(is-path v (x#xs) v' \wedge (x1,x2,x3) \neq (x3,x2,x1) \wedge (x3,x2,x1) \notin set xs

\wedge distinct (x#xs) \wedge (set xs \cap set (rev-path xs)={\}))

by (metis (full-types) distinct.simps(2) x)

also have ...=(is-path v (x#xs) v' \wedge (x1,x2,x3) \neq (x3,x2,x1) \wedge distinct (x#xs)

\wedge (x3,x2,x1) \notin set xs \wedge set xs \cap set (rev-path (x#xs))={\})

proof –

have set (rev-path (x#xs)) = set ((rev-path xs)@[x3,x2,x1]) using x by auto

also have ... = set (rev-path xs) \cup {(x3,x2,x1)} by auto

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

thus ?thesis by blast

qed

also have ...=(is-path v (x#xs) v' \wedge distinct (x#xs) \wedge (set (x#xs) \cap set (rev-path (x#xs))={\}))

proof –

have (x3,x2,x1) \notin set xs \longleftrightarrow (x1,x2,x3) \notin set (rev-path xs) using in-set-rev-path by auto

moreover have $set (rev\text{-}path (x\#xs)) = set (rev\text{-}path xs) \cup \{(x3, x2, x1)\}$
unfolding $rev\text{-}path\text{-}def$ **using** x **by** $auto$
ultimately have $(x1, x2, x3) \neq (x3, x2, x1) \wedge (x3, x2, x1) \notin set xs$
 $\longleftrightarrow (x1, x2, x3) \notin set (rev\text{-}path (x\#xs))$ **by** $blast$
thus $?thesis$
by $(metis (mono\text{-}tags) Int\text{-}iff Int\text{-}insert\text{-}left\text{-}if0 List.set\text{-}simps(2) empty\text{-}iff insertI1 x)$
qed
finally have $is\text{-}trail v (x\#xs) v' \longleftrightarrow (is\text{-}path v (x\#xs) v' \wedge distinct (x\#xs)$
 $\wedge (set (x\#xs) \cap set (rev\text{-}path (x\#xs)) = \{\}))$.
thus $?case$.
qed

lemma $(in\ valid\text{-}unMultigraph)$ $is\text{-}trail\text{-}rev$:
 $is\text{-}trail v' (rev\text{-}path ps) v \longleftrightarrow is\text{-}trail v ps v'$
using $rev\text{-}path\text{-}append is\text{-}trail\text{-}path is\text{-}path\text{-}rev distinct\text{-}rev\text{-}path$
by $(metis Int\text{-}commute distinct\text{-}append)$

lemma $(in\ valid\text{-}unMultigraph)$ $is\text{-}trail\text{-}intro[intro]$:
 $is\text{-}trail v' ps v \implies is\text{-}path v' ps v$ **by** $(induct ps arbitrary: v', auto)$

lemma $(in\ valid\text{-}unMultigraph)$ $is\text{-}trail\text{-}split$:
 $is\text{-}trail v (p1 @ p2) v' \implies (\exists u. is\text{-}trail v p1 u \wedge is\text{-}trail u p2 v')$
apply $(induct p1 arbitrary: v, auto)$
apply $(metis is\text{-}trail\text{-}intro is\text{-}path\text{-}memb)$
done

lemma $(in\ valid\text{-}unMultigraph)$ $is\text{-}trail\text{-}split'$: $is\text{-}trail v (p1 @ (u, w, u') \# p2) v'$
 $\implies is\text{-}trail v p1 u \wedge (u, w, u') \in E \wedge is\text{-}trail u' p2 v'$
by $(metis is\text{-}trail.simps(2) is\text{-}trail\text{-}split)$

lemma $(in\ valid\text{-}unMultigraph)$ $distinct\text{-}elim[simp]$:
assumes $is\text{-}trail v ((v1, w, v2) \# ps) v'$
shows $(v1, w, v2) \in edges (rem\text{-}unPath ps G) \longleftrightarrow (v1, w, v2) \in E$

proof
assume $(v1, w, v2) \in edges (rem\text{-}unPath ps G)$
thus $(v1, w, v2) \in E$ **by** $(metis assms is\text{-}trail.simps(2))$
next
assume $(v1, w, v2) \in E$
have $(v1, w, v2) \notin set ps \wedge (v2, w, v1) \notin set ps$ **by** $(metis assms is\text{-}trail.simps(2))$
hence $(v1, w, v2) \notin set ps \wedge (v1, w, v2) \notin set (rev\text{-}path ps)$ **by** $simp$
hence $(v1, w, v2) \notin set ps \cup set (rev\text{-}path ps)$ **by** $simp$
hence $(v1, w, v2) \in edges G - (set ps \cup set (rev\text{-}path ps))$
using $\langle (v1, w, v2) \in E \rangle$ **by** $auto$
thus $(v1, w, v2) \in edges (rem\text{-}unPath ps G)$
by $(metis rem\text{-}unPath\text{-}edges)$
qed

lemma $distinct\text{-}path\text{-}subset$:

assumes *valid-unMultigraph* $G1$ *valid-unMultigraph* $G2$ *edges* $G1 \subseteq \text{edges } G2$
nodes $G1 \subseteq \text{nodes } G2$
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 $v=v' \wedge v' \in \text{nodes } G1$
by (*metis (full-types) assms(1) valid-unMultigraph.is-trail.simps(1)*)
hence $v=v' \wedge v' \in \text{nodes } G2$ **using** $\langle \text{nodes } G1 \subseteq \text{nodes } G2 \rangle$ **by** *auto*
thus *?case* **by** (*metis assms(2) valid-unMultigraph.is-trail.simps(1)*)
next
case (*Cons x xs*)
obtain $x1$ $x2$ $x3$ **where** $x:x=(x1,x2,x3)$ **by** (*metis prod-cases3*)
hence *valid-unMultigraph.is-trail* $G1$ $x3$ xs v'
by (*metis Cons.premss assms(1) valid-unMultigraph.is-trail.simps(2)*)
hence *valid-unMultigraph.is-trail* $G2$ $x3$ xs v' **using** *Cons* **by** *auto*
moreover **have** $x \in \text{edges } G1$
by (*metis Cons.premss assms(1) valid-unMultigraph.is-trail.simps(2) x*)
hence $x \in \text{edges } G2$ **using** $\langle \text{edges } G1 \subseteq \text{edges } G2 \rangle$ **by** *auto*
moreover **have** $v=x1 \wedge (x1,x2,x3) \notin \text{set } xs \wedge (x3,x2,x1) \notin \text{set } xs$
by (*metis Cons.premss assms(1) valid-unMultigraph.is-trail.simps(2) x*)
hence $v=x1$ $(x1,x2,x3) \notin \text{set } xs$ $(x3,x2,x1) \notin \text{set } xs$ **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) $\subseteq V$ **by** *auto*
moreover **have** *edges* (*rem-unPath* p G) $\subseteq E$
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 (*in valid-unMultigraph*) *distinct-elim-rev[simp]*:
assumes *is-trail* v $((v1,w,v2)\#ps)$ v'

shows $(v2, w, v1) \in \text{edges}(\text{rem-unPath } ps \ G) \longleftrightarrow (v2, w, v1) \in E$
proof –
have $\text{valid-unMultigraph } (\text{rem-unPath } ps \ G)$ **using** $\text{valid-unMultigraph-axioms}$
by auto
hence $(v2, w, v1) \in \text{edges}(\text{rem-unPath } ps \ G) \longleftrightarrow (v1, w, v2) \in \text{edges}(\text{rem-unPath } ps \ G)$
by $(\text{metis } \text{valid-unMultigraph.corres})$
moreover have $(v2, w, v1) \in E \longleftrightarrow (v1, w, v2) \in E$ **using** corres **by simp**
ultimately show $?thesis$ **using** distinct-elim **by** $(\text{metis } \text{assms})$
qed

lemma $(\text{in } \text{valid-unMultigraph}) \text{ del-UnEdge-even}$:
assumes $(v, w, v') \in E$ $\text{finite } E$
shows $v \in \text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G) \longleftrightarrow \text{even } (\text{degree } v \ G)$
proof –
have $\text{degree } v \ (\text{del-unEdge } v \ w \ v' \ G) + 1 = \text{degree } v \ G$
using $\text{del-edge-undirected-degree-plus}$ **corres** **by** $(\text{metis } \text{assms})$
from this $[\text{symmetric}]$ **have** $\text{odd } (\text{degree } v \ (\text{del-unEdge } v \ w \ v' \ G)) = \text{even } (\text{degree } v \ G)$
by simp
moreover have $v \in \text{nodes } (\text{del-unEdge } v \ w \ v' \ G)$ **by** $(\text{metis } E\text{-validD}(1) \ \text{assms}(1) \ \text{del-UnEdge-node})$
ultimately show $?thesis$ **unfolding** odd-nodes-set-def **by auto**
qed

lemma $(\text{in } \text{valid-unMultigraph}) \text{ del-UnEdge-even}'$:
assumes $(v, w, v') \in E$ $\text{finite } E$
shows $v' \in \text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G) \longleftrightarrow \text{even } (\text{degree } v' \ G)$
proof –
show $?thesis$ **by** $(\text{metis } (\text{full-types}) \ \text{assms} \ \text{corres} \ \text{del-UnEdge-even} \ \text{delete-edge-sym})$

qed

lemma $\text{del-UnEdge-even-even}$:
assumes $\text{valid-unMultigraph } G$ $\text{finite}(\text{edges } G)$ $\text{finite}(\text{nodes } G)$ $(v, w, v') \in \text{edges } G$
assumes $\text{parity-assms: even } (\text{degree } v \ G)$ $\text{even } (\text{degree } v' \ G)$
shows $\text{num-of-odd-nodes}(\text{del-unEdge } v \ w \ v' \ G) = \text{num-of-odd-nodes } G + 2$
proof –
interpret $G: \text{valid-unMultigraph}$ **by fact**
have $v \in \text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G)$
by $(\text{metis } G.\text{del-UnEdge-even} \ \text{assms}(2) \ \text{assms}(4) \ \text{parity-assms}(1))$
moreover have $v' \in \text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G)$
by $(\text{metis } G.\text{del-UnEdge-even}' \ \text{assms}(2) \ \text{assms}(4) \ \text{parity-assms}(2))$
ultimately have $\text{extra-odd-nodes: } \{v, v'\} \subseteq \text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G)$
unfolding odd-nodes-set-def **by auto**
moreover have $v \notin \text{odd-nodes-set } G$ **and** $v' \notin \text{odd-nodes-set } G$
using parity-assms **unfolding** odd-nodes-set-def **by auto**
hence $v v'\text{-odd-disjoint: } \{v, v'\} \cap \text{odd-nodes-set } G = \{\}$ **by auto**

moreover have $odd\text{-nodes-set}(del\text{-unEdge } v \ w \ v' \ G) - \{v, v'\} \subseteq odd\text{-nodes-set } G$
proof
fix x
assume $x\text{-odd-set}: x \in odd\text{-nodes-set } (del\text{-unEdge } v \ w \ v' \ G) - \{v, v'\}$
hence $degree \ x \ (del\text{-unEdge } v \ w \ v' \ G) = degree \ x \ G$
by (*metis Diff-iff G.degree-frame assms(2)*)
hence $odd(degrees \ x \ G)$ **using** $x\text{-odd-set}$
unfolding $odd\text{-nodes-set-def}$ **by** *auto*
moreover have $x \in nodes \ G$ **using** $x\text{-odd-set}$ **unfolding** $odd\text{-nodes-set-def}$
by *auto*
ultimately show $x \in odd\text{-nodes-set } G$ **unfolding** $odd\text{-nodes-set-def}$ **by** *auto*
qed
moreover have $odd\text{-nodes-set } G \subseteq odd\text{-nodes-set}(del\text{-unEdge } v \ w \ v' \ G)$
proof
fix x
assume $x\text{-odd-set}: x \in odd\text{-nodes-set } G$
hence $x \notin \{v, v'\} \implies odd(degrees \ x \ (del\text{-unEdge } v \ w \ v' \ G))$
by (*metis (lifting) G.degree-frame assms(2) mem-Collect-eq odd-nodes-set-def*)
hence $x \notin \{v, v'\} \implies x \in odd\text{-nodes-set}(del\text{-unEdge } v \ w \ v' \ G)$
using $x\text{-odd-set}$ $del\text{-UnEdge-node}$ **unfolding** $odd\text{-nodes-set-def}$ **by** *auto*
moreover have $x \in \{v, v'\} \implies x \in odd\text{-nodes-set}(del\text{-unEdge } v \ w \ v' \ G)$
using $extra\text{-odd-nodes}$ **by** *auto*
ultimately show $x \in odd\text{-nodes-set } (del\text{-unEdge } v \ w \ v' \ G)$ **by** *auto*
qed
ultimately have $odd\text{-nodes-set}(del\text{-unEdge } v \ w \ v' \ G) = odd\text{-nodes-set } G \cup \{v, v'\}$
by *auto*
thus $num\text{-of-odd-nodes}(del\text{-unEdge } v \ w \ v' \ G) = num\text{-of-odd-nodes } G + 2$
proof –
assume $odd\text{-nodes-set}(del\text{-unEdge } v \ w \ v' \ G) = odd\text{-nodes-set } G \cup \{v, v'\}$
moreover have $v \neq v'$ **using** $G.no\text{-id } \langle (v, w, v') \in edges \ G \rangle$ **by** *auto*
hence $card\{v, v'\} = 2$ **by** *simp*
moreover have $odd\text{-nodes-set } G \cap \{v, v'\} = \{\}$
using $vv'\text{-odd-disjoint}$ **by** *auto*
moreover have $finite(odd\text{-nodes-set } G)$
by (*metis (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** $num\text{-of-odd-nodes-def}$ **using** $card\text{-Un-disjoint}$
by *metis*
qed
qed

lemma $del\text{-UnEdge-even-odd}$:
assumes $valid\text{-unMultigraph } G \ finite(edges \ G) \ finite(nodes \ G) \ (v, w, v') \in edges \ G$
assumes $parity\text{-assms}: even \ (degree \ v \ G) \ odd \ (degree \ v' \ G)$
shows $num\text{-of-odd-nodes}(del\text{-unEdge } v \ w \ v' \ G) = num\text{-of-odd-nodes } G$
proof –
interpret $G : valid\text{-unMultigraph}$ **by** *fact*

```

have odd-v:v∈odd-nodes-set(del-unEdge v w v' G)
  by (metis G.del-UnEdge-even assms(2) assms(4) parity-assms(1))
have not-odd-v':v'∉odd-nodes-set(del-unEdge v w v' G)
  by (metis G.del-UnEdge-even' assms(2) assms(4) parity-assms(2))
have odd-nodes-set(del-unEdge v w v' G) ∪ {v'} ⊆ odd-nodes-set G ∪ {v}
proof
  fix x
  assume x-prems: x ∈ odd-nodes-set (del-unEdge v w v' G) ∪ {v'}
  have x=v' ⇒ x∈odd-nodes-set G ∪ {v}
    using parity-assms
  by (metis (lifting) G.E-validD(2) Un-def assms(4) mem-Collect-eq odd-nodes-set-def
)
  moreover have x=v ⇒ x∈odd-nodes-set G ∪ {v}
    by (metis insertI1 insert-is-Un sup-commute)
  moreover have x∉{v,v'} ⇒ x ∈ odd-nodes-set (del-unEdge v w v' G)
    using x-prems by auto
  hence x∉{v,v'} ⇒ x ∈ odd-nodes-set G unfolding odd-nodes-set-def
    using G.degree-frame ⟨finite (edges G)⟩ by auto
  hence x∉{v,v'} ⇒ x∈odd-nodes-set G ∪ {v} by simp
  ultimately show x ∈ odd-nodes-set G ∪ {v} by auto
qed
moreover have odd-nodes-set G ∪ {v} ⊆ odd-nodes-set(del-unEdge v w v' G) ∪
{v'}
proof
  fix x
  assume x-prems: x ∈ odd-nodes-set G ∪ {v}
  have x=v ⇒ x ∈ odd-nodes-set (del-unEdge v w v' G) ∪ {v'}
    by (metis UnI1 odd-v)
  moreover have x=v' ⇒ x ∈ odd-nodes-set (del-unEdge v w v' G) ∪ {v'}
    by auto
  moreover have x∉{v,v'} ⇒ x ∈ odd-nodes-set G ∪ {v} using x-prems by
auto
  hence x∉{v,v'} ⇒ x∈odd-nodes-set (del-unEdge v w v' G) unfolding
odd-nodes-set-def
    using G.degree-frame ⟨finite (edges G)⟩ by auto
  hence x∉{v,v'} ⇒ x ∈ odd-nodes-set (del-unEdge v w v' G) ∪ {v'} by simp
  ultimately show x ∈ odd-nodes-set (del-unEdge v w v' G) ∪ {v'} by auto
qed
ultimately have odd-nodes-set(del-unEdge v w v' G) ∪ {v'} = odd-nodes-set G
∪ {v}
  by auto
moreover have odd-nodes-set G ∩ {v} = {}
  using parity-assms unfolding odd-nodes-set-def by auto
moreover have odd-nodes-set(del-unEdge v w v' G) ∩ {v'} = {}
  by (metis Int-insert-left-if0 inf-bot-left inf-commute not-odd-v')
moreover have finite (odd-nodes-set(del-unEdge v w v' G))
  using ⟨finite (nodes G)⟩ by auto
moreover have finite (odd-nodes-set G) using ⟨finite (nodes G)⟩ by auto
ultimately have card(odd-nodes-set G) + card {v} =

```

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

lemma *del-UnEdge-odd-even*:
assumes *valid-unMultigraph G finite(edges G) finite(nodes G) (v, w, v') ∈ edges G*
assumes *parity-assms: odd (degree v G) even (degree v' G)*
shows $\text{num-of-odd-nodes}(\text{del-unEdge } v \ w \ v' \ G) = \text{num-of-odd-nodes } G$
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') ∈ edges G*
assumes *parity-assms: odd (degree v G) odd (degree v' G)*
shows $\text{num-of-odd-nodes } G = \text{num-of-odd-nodes}(\text{del-unEdge } v \ w \ v' \ G) + 2$
proof –
interpret *G:valid-unMultigraph by fact*
have $v \notin \text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G)$
by (*metis G.del-UnEdge-even assms(2) assms(4) parity-assms(1)*)
moreover have $v' \notin \text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G)$
by (*metis G.del-UnEdge-even' assms(2) assms(4) parity-assms(2)*)
ultimately have $v \ v' \text{-disjoint: } \{v, v'\} \cap \text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G) = \{\}$

by (*metis (full-types) Int-insert-left-if0 inf-bot-left*)
moreover have $\text{extra-odd-nodes: } \{v, v'\} \subseteq \text{odd-nodes-set } G$
unfolding *odd-nodes-set-def*
using $\langle (v, w, v') \in \text{edges } G \rangle$
by (*metis (lifting) G.E-validD empty-subsetI insert-subset mem-Collect-eq parity-assms*)
moreover have $\text{odd-nodes-set } G - \{v, v'\} \subseteq \text{odd-nodes-set } (\text{del-unEdge } v \ w \ v' \ G)$

proof
fix *x*
assume $x \in \text{odd-nodes-set } G - \{v, v'\}$
hence $\text{degree } x \ G = \text{degree } x \ (\text{del-unEdge } v \ w \ v' \ G)$
by (*metis Diff-iff G.degree-frame assms(2)*)
hence $\text{odd}(\text{degree } x \ (\text{del-unEdge } v \ w \ v' \ G))$ **using** *x-odd-set*
unfolding *odd-nodes-set-def by auto*
moreover have $x \in \text{nodes } (\text{del-unEdge } v \ w \ v' \ G)$
using *x-odd-set unfolding odd-nodes-set-def by auto*
ultimately show $x \in \text{odd-nodes-set } (\text{del-unEdge } v \ w \ v' \ G)$
unfolding *odd-nodes-set-def by auto*
qed

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

```

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

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

lemma (in valid-unMultigraph) 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 ?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  $\text{rem-x:odd } (\text{degree } v' \ (\text{rem-unPath } (x\#xs) \ G)) = \text{odd}(\text{degree } v' \ (\text{del-unEdge } x1 \ x2 \ x3 \ (\text{rem-unPath } xs \ G)))$ 
  by (metis rem-unPath.simps(2) rem-unPath-com)
  have  $x3 = v' \implies ?case$ 

```



```

proof (cases v=v')
  case True
  assume x3=v'
  have x1=v' using x by (metis Cons.prem(2) True is-trail.simp(2))
  thus ?thesis using ⟨x3=v'⟩ by (metis Cons.prem(2) is-trail.simp(2) no-id
x)
next
  case False
  assume x3=v'
  have odd (degree v' (rem-unPath (x # xs) G)) = odd (degree v' (
    del-unEdge x1 x2 x3 (rem-unPath xs G))) using rem-x .
  also have ...=odd (degree v' (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 (x1,x2,x3) ∈ edges (rem-unPath xs G)
      by (metis Cons.prem(2) distinct-elim is-trail.simp(2) x)
    moreover have (x3,x2,x1) ∈ edges (rem-unPath xs G)
      by (metis Cons.prem(2) corres distinct-elim-rev is-trail.simp(2) x)
    ultimately show ?thesis
      by (metis ⟨x3 = v'⟩ del-edge-undirected-degree-minus delete-edge-sym x)
  qed
  also have ...=even (degree v' (rem-unPath xs G))
  proof -
    have (x1,x2,x3) ∈ E by (metis Cons.prem(2) is-trail.simp(2) x)
    hence (x3,x2,x1) ∈ edges (rem-unPath xs G)
      by (metis Cons.prem(2) corres distinct-elim-rev x)
    hence (x3,x2,x1) ∈ {e ∈ edges (rem-unPath xs G). fst e = v'}
      using ⟨x3=v'⟩ by (metis (mono-tags) fst-conv mem-Collect-eq)
    moreover have finite {e ∈ edges (rem-unPath xs G). fst e = v'}
      using ⟨finite E⟩ by auto
    ultimately have degree v' (rem-unPath xs G) ≠ 0
      unfolding degree-def by auto
    thus ?thesis by auto
  qed
  also have ...=even (degree v' G)
    using ⟨x3 = v'⟩ assms
    by (metis (mono-tags) Cons.hyps Cons.prem(2) is-trail.simp(2) x)
  finally have odd (degree v' (rem-unPath (x # xs) G)) = even (degree v' G) .
  thus ?thesis by (metis False)
qed
moreover have x3 ≠ v' ⇒ ?case
  proof (cases v=v')
    case True
    assume x3=v'
    have odd (degree v' (rem-unPath (x # xs) G)) = odd (degree v' (
      del-unEdge x1 x2 x3 (rem-unPath xs G))) using rem-x .
    also have ...=odd (degree v' (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 (x1,x2,x3) ∈ edges (rem-unPath xs G)
  by (metis Cons.prem(2) distinct-elim is-trail.simps(2) x)
moreover have (x3,x2,x1) ∈ edges (rem-unPath xs G)
  by (metis Cons.prem(2) corres distinct-elim-rev is-trail.simps(2) x)
ultimately show ?thesis
  using True x
by (metis Cons.prem(2) del-edge-undirected-degree-minus is-trail.simps(2))
qed
also have ...=even (degree v' (rem-unPath xs G))
proof –
  have (x1,x2,x3) ∈ E by (metis Cons.prem(2) is-trail.simps(2) x)
  hence (x1,x2,x3) ∈ edges (rem-unPath xs G)
    by (metis Cons.prem(2) distinct-elim x)
  hence (x1,x2,x3) ∈ {e ∈ edges (rem-unPath xs G). fst e = v'}
    using ⟨v=v'⟩ x Cons
    by (metis (lifting, mono-tags) fst-conv is-trail.simps(2) mem-Collect-eq)
  moreover have finite {e ∈ edges (rem-unPath xs G). fst e = v'}
    using ⟨finite E⟩ by auto
  ultimately have degree v' (rem-unPath xs G) ≠ 0
    unfolding degree-def by auto
  thus ?thesis by auto
qed
also have ...≠even (degree v' G)
  using ⟨x3 ≠ v'⟩ assms
  by (metis Cons.hyps Cons.prem(2) is-trail.simps(2) x)
finally have odd (degree v' (rem-unPath (x # xs) G)) ≠ even (degree v' G) .
thus ?thesis by (metis True)
next
case False
assume x3 ≠ v'
have odd (degree v' (rem-unPath (x # xs) G)) = odd (degree v' (
  del-unEdge x1 x2 x3 (rem-unPath xs G))) using rem-x .
also have ...=odd (degree v' (rem-unPath xs G))
proof –
  have v=x1 by (metis Cons.prem(2) is-trail.simps(2) x)
  hence v' ∉ {x1,x3} by (metis (mono-tags) False ⟨x3 ≠ v'⟩ empty-iff
insert-iff)
  moreover have valid-unMultigraph (rem-unPath xs G)
    using valid-unMultigraph-axioms by auto
  moreover have finite (edges (rem-unPath xs G))
    by (metis (full-types) assms(1) finite-Diff rem-unPath-edges)
  ultimately have degree v' (del-unEdge x1 x2 x3 (rem-unPath xs G))
    = degree v' (rem-unPath xs G) using degree-frame
    by (metis valid-unMultigraph.degree-frame)
  thus ?thesis by simp
qed
also have ...=even (degree v' G)

```

using *assms* $x \langle x3 \neq v' \rangle$
by (*metis Cons.hyps Cons.premis(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' rem-unPath-graph*)

lemma (*in valid-unMultigraph*) *rem-UnPath-parity-others*:
assumes *finite E is-trail v ps v' n \notin {v, v'}*
shows $even (degree\ n (rem\ unPath\ ps\ G)) = even (degree\ n\ G)$ **using** *assms*
proof (*induct ps arbitrary: v*)
case *Nil*
thus *?case* **by** *auto*

next
case (*Cons x xs*)
obtain $x1\ x2\ x3$ **where** $x : x = (x1, x2, x3)$ **by** (*metis prod-cases3*)
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 $\dots = even (degree\ n (rem\ unPath\ xs\ G) - 1)$
proof –
have *finite (edges (rem-unPath xs G))*
by (*metis (full-types) assms(1) finite-Diff rem-unPath-edges*)
moreover have $(x1, x2, x3) \in edges (rem\ unPath\ xs\ G)$
by (*metis Cons.premis(2) distinct-elim is-trail.simps(2) x*)
moreover have $(x3, x2, x1) \in edges (rem\ unPath\ xs\ G)$
by (*metis Cons.premis(2) corres distinct-elim-rev is-trail.simps(2) x*)
ultimately show *?thesis*
using $\langle n = x3 \rangle del\ edge\ undirected\ degree\ minus'$
by *auto*

qed
also have $\dots = odd (degree\ n (rem\ unPath\ xs\ G))$
proof –
have $(x1, x2, x3) \in E$ **by** (*metis Cons.premis(2) is-trail.simps(2) x*)
hence $(x3, x2, x1) \in edges (rem\ unPath\ xs\ G)$
by (*metis Cons.premis(2) corres distinct-elim-rev x*)
hence $(x3, x2, x1) \in \{e \in edges (rem\ unPath\ xs\ G). fst\ e = n\}$

```

    using ⟨n=x3⟩ by (metis (mono-tags) fst-conv mem-Collect-eq)
  moreover have finite {e ∈ edges (rem-unPath xs G). fst e = n}
    using ⟨finite E⟩ by auto
  ultimately have degree n (rem-unPath xs G) ≠ 0
    unfolding degree-def by auto
  thus ?thesis by auto
qed
also have ...=even(degree n G)
proof -
  have x3 ≠ v' by (metis ⟨n = x3⟩ assms(3) insert-iff)
  hence odd (degree x3 (rem-unPath xs G)) = even(degree x3 G)
    using Cons assms
  by (metis is-trail.simps(2) rem-UnPath-parity-v x)
  thus ?thesis using ⟨n=x3⟩ by auto
qed
finally have even (degree n (rem-unPath (x#xs) G))=even(degree n G) .
thus ?thesis .
qed
moreover have n ≠ x3 ⇒ ?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 ...=even(degree n (rem-unPath xs G))
  proof -
    have v=x1 by (metis Cons.prem(2) is-trail.simps(2) x)
    hence n ∉ {x1,x3} by (metis Cons.prem(3) ⟨n ≠ x3⟩ insertE insertI1
singletonE)
  moreover have valid-unMultigraph (rem-unPath xs G)
    using valid-unMultigraph-axioms by auto
  moreover have finite (edges (rem-unPath xs G))
    by (metis (full-types) assms(1) finite-Diff rem-unPath-edges)
  ultimately have degree n (del-unEdge x1 x2 x3 (rem-unPath xs G))
    =degree n (rem-unPath xs G) using degree-frame
  by (metis valid-unMultigraph.degree-frame)
  thus ?thesis by simp
qed
also have ...=even(degree n G)
  using Cons assms ⟨n ≠ x3⟩ x by auto
finally have even (degree n (rem-unPath (x#xs) G))=even(degree n G) .
thus ?thesis .
qed
ultimately show ?case by auto
qed

lemma (in valid-unMultigraph) rem-UnPath-even:
  assumes finite E finite V is-trail v ps v'
  assumes parity-assms: even (degree v' G)

```

shows $\text{num-of-odd-nodes } (\text{rem-unPath } ps \ G) = \text{num-of-odd-nodes } G$
 $+ (\text{if even } (\text{degree } v \ G) \wedge v \neq v' \text{ then } 2 \text{ else } 0)$ **using** *assms*

proof (*induct ps arbitrary:v*)

case *Nil*

thus *?case by auto*

next

case (*Cons x xs*)

obtain $x1 \ x2 \ x3$ **where** $x:x=(x1,x2,x3)$ **by** (*metis prod-cases3*)

have *fin-nodes: finite (nodes (rem-unPath xs G))* **using** *Cons by auto*

have *fin-edges: finite (edges (rem-unPath xs G))* **using** *Cons by auto*

have *valid-rem-xs: valid-unMultigraph (rem-unPath xs G)* **using** *valid-unMultigraph-axioms*

by *auto*

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

by (*metis (full-types) Cons.prem3 distinct-elim is-trail.simps(2) x*)

have $\text{even } (\text{degree } x1 \ (\text{rem-unPath } xs \ G))$
 $\implies \text{even}(\text{degree } x3 \ (\text{rem-unPath } xs \ G)) \implies ?\text{case}$

proof –

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

have $\text{num-of-odd-nodes } (\text{rem-unPath } (x\#xs) \ G) = \text{num-of-odd-nodes}$
 $(\text{del-unEdge } x1 \ x2 \ x3 \ (\text{rem-unPath } xs \ G))$

by (*metis rem-unPath.simps(2) rem-unPath-com x*)

also have $\dots = \text{num-of-odd-nodes } (\text{rem-unPath } xs \ G) + 2$

using *parity-x1-x3 fin-nodes fin-edges valid-rem-xs x-in del-UnEdge-even-even*

by *metis*

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

using *Cons.hyps[OF ‹finite E› ‹finite V›, of x3] ‹is-trail v (x # xs) v›*
 $\langle \text{even } (\text{degree } v' \ G) \rangle x$

by *auto*

also have $\dots = \text{num-of-odd-nodes } G + 2$

proof –

have $\text{even}(\text{degree } x3 \ G) \wedge x3 \neq v' \iff \text{odd } (\text{degree } x3 \ (\text{rem-unPath } xs \ G))$

using *Cons.prem3 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 $\dots = \text{num-of-odd-nodes } G + (\text{if even}(\text{degree } v \ G) \wedge v \neq v' \text{ then } 2 \text{ else } 0)$

proof –

have $x1 \neq x3$ **by** (*metis valid-rem-xs valid-unMultigraph.no-id x-in*)

moreover hence $x1 \neq v'$

using *Cons assms*

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

ultimately have $x1 \notin \{x3, v'\}$ **by** *auto*

hence $\text{even}(\text{degree } x1 \ G)$

using *Cons.prem3(3) assms(1) assms(2) parity-x1-x3(1)*

by (metis (full-types) is-trail.simps(2) rem-UnPath-parity-others x)
 hence even(degrees x1 G) \wedge $x1 \neq v'$ using $\langle x1 \neq v' \rangle$ by auto
 hence even(degrees v G) \wedge $v \neq v'$ by (metis Cons.prem(3) is-trail.simps(2))

x)

thus ?thesis by auto
 qed

finally have num-of-odd-nodes (rem-unPath (x#xs) G) =
 num-of-odd-nodes G + (if even(degrees v G) \wedge $v \neq v'$ then 2 else

0) .

thus ?thesis .
 qed

moreover have even (degrees x1 (rem-unPath xs G)) \implies
 odd(degrees x3 (rem-unPath xs G)) \implies ?case

proof –

assume parity-x1-x3: even (degrees x1 (rem-unPath xs G))
 odd (degrees 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-even-odd)

also have ... = num-of-odd-nodes G + (if even(degrees x3 G) \wedge $x3 \neq v'$ then 2 else

0)

using Cons.hyps Cons.prem(3) assms(1) assms(2) parity-assms x
 by auto

also have ... = num-of-odd-nodes G + 2

proof –

have even(degrees x3 G) \wedge $x3 \neq v' \iff$ odd (degrees x3 (rem-unPath xs G))
 using Cons.prem 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(degrees v G) \wedge $v \neq v'$ then 2 else

0)

proof –

have $x1 \neq x3$ by (metis valid-rem-xs valid-unMultigraph.no-id x-in)
 moreover hence $x1 \neq v'$
 using Cons assms
 by (metis is-trail.simps(2) parity-x1-x3(1) rem-UnPath-parity-v' x)
 ultimately have $x1 \notin \{x3, v'\}$ by auto
 hence even(degrees x1 G)
 using Cons.prem(3) assms(1) assms(2) parity-x1-x3(1)
 by (metis (full-types) is-trail.simps(2) rem-UnPath-parity-others x)
 hence even(degrees x1 G) \wedge $x1 \neq v'$ using $\langle x1 \neq v' \rangle$ by auto
 hence even(degrees v G) \wedge $v \neq v'$ by (metis Cons.prem(3) is-trail.simps(2))

x)

thus ?thesis by auto
 qed

finally have $\text{num-of-odd-nodes } (\text{rem-unPath } (x\#xs) G) =$
 $\text{num-of-odd-nodes } G + (\text{if even}(\text{degree } v G) \wedge v \neq v' \text{ then } 2 \text{ else}$
0) .
thus ?thesis .
qed
moreover have $\text{odd } (\text{degree } x1 (\text{rem-unPath } xs G)) \implies$
 $\text{even}(\text{degree } x3 (\text{rem-unPath } xs G)) \implies ?\text{case}$
proof –
assume *parity-x1-x3*: $\text{odd } (\text{degree } x1 (\text{rem-unPath } xs G))$
 $\text{even } (\text{degree } x3 (\text{rem-unPath } xs G))$
have $\text{num-of-odd-nodes } (\text{rem-unPath } (x\#xs) G) = \text{num-of-odd-nodes}$
 $(\text{del-unEdge } x1 x2 x3 (\text{rem-unPath } xs G))$
by (*metis rem-unPath.simps(2) rem-unPath-com x*)
also have $\dots = \text{num-of-odd-nodes } (\text{rem-unPath } xs G)$
using *parity-x1-x3 fin-nodes fin-edges valid-rem-xs x-in*
by (*metis del-UnEdge-odd-even*)
also have $\dots = \text{num-of-odd-nodes } G + (\text{if even}(\text{degree } x3 G) \wedge x3 \neq v' \text{ then } 2 \text{ else}$
0))
using *Cons.hyps Cons.prem(3) assms(1) assms(2) parity-assms x*
by *auto*
also have $\dots = \text{num-of-odd-nodes } G$
proof –
have $\text{even}(\text{degree } x3 G) \wedge x3 \neq v' \iff \text{odd } (\text{degree } x3 (\text{rem-unPath } xs G))$
using *Cons.prem(3) 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 $\dots = \text{num-of-odd-nodes } G + (\text{if even}(\text{degree } v G) \wedge v \neq v' \text{ then } 2 \text{ else}$
0))
proof (*cases v ≠ 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.prem(3) is-trail.simps(2) x*)
ultimately have $\text{odd } (\text{degree } x1 (\text{rem-unPath } xs G))$
 $\iff \text{odd}(\text{degree } x1 G)$
using *True parity-x1-x3(1) rem-UnPath-parity-others x Cons.prem(3)*
assms(1) assms(2)
by *auto*
hence $\text{odd}(\text{degree } x1 G)$ **by** (*metis parity-x1-x3(1)*)
thus ?thesis
by (*metis (mono-tags) Cons.prem(3) Nat.add-0-right is-trail.simps(2)*)
x)
next
case *False*
then show ?thesis **by** *auto*
qed
finally have $\text{num-of-odd-nodes } (\text{rem-unPath } (x\#xs) G) =$
 $\text{num-of-odd-nodes } G + (\text{if even}(\text{degree } v G) \wedge v \neq v' \text{ then } 2 \text{ else}$

$0)$.
thus *?thesis* .
qed
moreover have $\text{odd}(\text{degree } x1 \text{ (rem-unPath } xs \ G)) \implies$
 $\text{odd}(\text{degree } x3 \text{ (rem-unPath } xs \ G)) \implies \text{?case}$
proof –
assume *parity-x1-x3*: $\text{odd}(\text{degree } x1 \text{ (rem-unPath } xs \ G))$
 $\text{odd}(\text{degree } x3 \text{ (rem-unPath } xs \ G))$
have $\text{num-of-odd-nodes}(\text{rem-unPath}(x\#xs) \ G) = \text{num-of-odd-nodes}$
 $(\text{del-unEdge } x1 \ x2 \ x3 \text{ (rem-unPath } xs \ G))$
by (*metis rem-unPath.simps(2) rem-unPath-com x*)
also have $\dots = \text{num-of-odd-nodes}(\text{rem-unPath } xs \ G) - (2::\text{nat})$
using *del-UnEdge-odd-odd*
by (*metis add-implies-diff fin-edges fin-nodes parity-x1-x3 valid-rem-xs x-in*)

also have $\dots = \text{num-of-odd-nodes } G + (\text{if even}(\text{degree } x3 \ G) \wedge x3 \neq v' \text{ then } 2 \text{ else}$
 $0) - (2::\text{nat})$
using *Cons assms*
by (*metis is-trail.simps(2) x*)
also have $\dots = \text{num-of-odd-nodes } G$
proof –
have $\text{even}(\text{degree } x3 \ G) \wedge x3 \neq v' \iff \text{odd}(\text{degree } x3 \text{ (rem-unPath } xs \ G))$
using *Cons.premss 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 $\dots = \text{num-of-odd-nodes } G + (\text{if even}(\text{degree } v \ G) \wedge v \neq v' \text{ then } 2 \text{ else}$
 $0)$
proof (*cases v ≠ 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.premss(3) is-trail.simps(2) x*)
ultimately have $\text{odd}(\text{degree } x1 \text{ (rem-unPath } xs \ G))$
 $\iff \text{odd}(\text{degree } x1 \ G)$
using *True Cons.premss(3) assms(1) assms(2) parity-x1-x3(1) rem-UnPath-parity-others*
 x
by *auto*
hence $\text{odd}(\text{degree } x1 \ G)$ **by** (*metis parity-x1-x3(1)*)
thus *?thesis*
by (*metis (mono-tags) Cons.premss(3) Nat.add-0-right is-trail.simps(2)*)
 $x)$
next
case *False*
thus *?thesis by (metis (mono-tags) add-0-iff)*
qed
finally have $\text{num-of-odd-nodes}(\text{rem-unPath}(x\#xs) \ G) =$
 $\text{num-of-odd-nodes } G + (\text{if even}(\text{degree } v \ G) \wedge v \neq v' \text{ then } 2 \text{ else}$
 $0)$.


```

    thus ?thesis .
  qed
  ultimately show ?case by metis
qed

lemma (in valid-unMultigraph) rem-UnPath-odd:
  assumes finite E finite V is-trail v ps v'
  assumes parity-assms: odd (degree v' G)
  shows num-of-odd-nodes (rem-unPath ps G) = num-of-odd-nodes G
    + (if odd (degree v G) ∧ v ≠ v' then -2 else 0) using assms
proof (induct ps arbitrary:v)
  case Nil
  thus ?case by auto
next
  case (Cons x xs)
  obtain x1 x2 x3 where x:x=(x1,x2,x3) by (metis prod-cases3)
  have fin-nodes: finite (nodes (rem-unPath xs G)) using Cons by auto
  have fin-edges: finite (edges (rem-unPath xs G)) using Cons by auto
  have valid-rem-xs: valid-unMultigraph (rem-unPath xs G) using valid-unMultigraph-axioms

  by auto
  have x-in:(x1,x2,x3)∈edges (rem-unPath xs G)
  by (metis (full-types) Cons.prem3 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.prem3 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 odd (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.prem3 is-trail.simp2 x)
  ultimately have even (degree x1 (rem-unPath xs G))
     $\longleftrightarrow$  even (degree x1 G)
    using True Cons.prem3 assms(1) assms(2) parity-x1-x3(1)
      rem-UnPath-parity-others x
    by auto
  hence even (degree x1 G) by (metis parity-x1-x3(1))
  thus ?thesis
    by (metis (opaque-lifting, mono-tags) Cons.prem3 is-trail.simp2)
      monoid-add-class.add.right-neutral x
  next
  case False
  then show ?thesis by auto
  qed
finally have num-of-odd-nodes (rem-unPath (x#xs) G)=
  num-of-odd-nodes G+(if odd(degree v G) ∧ v≠v' then -2 else
0) .
  thus ?thesis .
  qed
moreover have even (degree x1 (rem-unPath xs G))  $\implies$ 
  odd(degree x3 (rem-unPath xs G))  $\implies$  ?case
proof -
  assume parity-x1-x3: even (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.simp2 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-even-odd)
  also have ...=num-of-odd-nodes G+(if odd(degree x3 G) ∧ x3≠v' then - 2
else 0 )
  using Cons.hyps[OF ⟨finite E⟩ ⟨finite V⟩, of x3] Cons.prem3 assms(1)
  assms(2)
  parity-assms x
  by auto
  also have ...=num-of-odd-nodes G
  proof -
  have odd(degree x3 G) ∧ x3≠v'  $\longleftrightarrow$  even (degree x3 (rem-unPath xs G))
  using Cons.prem3 assms
  by (metis is-trail.simp2 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 odd(degree v G) ∧ v≠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.prem3 is-trail.simp2*) x

ultimately have *even* (*degree* $x1$ (*rem-unPath* xs G))

\longleftrightarrow *even* (*degree* $x1$ G)

using *True Cons.prem3 assms1 assms2 parity-x1-x31*

rem-UnPath-parity-others x

by *auto*

hence *even* (*degree* $x1$ G) **by** (*metis parity-x1-x31*)

with *Cons.prem3* x **show** *?thesis* **by** *auto*

next

case *False*

then show *?thesis* **by** *auto*

qed

finally have *num-of-odd-nodes* (*rem-unPath* ($x\#xs$) G) =

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

0) .

thus *?thesis* .

qed

moreover have *odd* (*degree* $x1$ (*rem-unPath* xs G)) \implies

even(*degree* $x3$ (*rem-unPath* xs G)) \implies *?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.simp2 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* $G + (\text{if } \text{odd}(\text{degree } x3 \ G) \wedge x3 \neq v' \text{ then } -2$

else 0)

using *Cons.hyps Cons.prem3 assms1 assms2 parity-assms* x

by *auto*

also have ... = *num-of-odd-nodes* $G + (-2)$

proof –

have $\text{odd}(\text{degree } x3 \ G) \wedge x3 \neq v' \longleftrightarrow$ *even* (*degree* $x3$ (*rem-unPath* xs G))

using *Cons.prem3 assms*

by (*metis is-trail.simp2 parity-x1-x32 rem-UnPath-parity-v*) x

hence $\text{odd}(\text{degree } x3 \ G) \wedge x3 \neq v'$ **by** (*metis parity-x1-x32*)

thus *?thesis* **by** *auto*

qed

also have ... = *num-of-odd-nodes* $G + (\text{if } \text{odd}(\text{degree } v \ G) \wedge v \neq v' \text{ then } -2 \text{ else}$

0)

proof –

have $x1 \neq x3$ **by** (*metis valid-rem-xs valid-unMultigraph.no-id x-in*)

moreover hence $x1 \neq v'$
using *Cons.assms*
by (*metis is-trail.simps(2) parity-x1-x3(1) rem-UnPath-parity-v' x*)
ultimately have $x1 \notin \{x3, v'\}$ **by** *auto*
hence $\text{odd}(\text{degree } x1 \ G)$
using *Cons.prem(3) assms(1) assms(2) parity-x1-x3(1)*
by (*metis (full-types) is-trail.simps(2) rem-UnPath-parity-others x*)
hence $\text{odd}(\text{degree } x1 \ G) \wedge x1 \neq v'$ **using** $\langle x1 \neq v' \rangle$ **by** *auto*
hence $\text{odd}(\text{degree } v \ G) \wedge v \neq v'$ **by** (*metis Cons.prem(3) is-trail.simps(2)*)
x)
thus *?thesis* **by** *auto*
qed
finally have $\text{num-of-odd-nodes } (\text{rem-unPath } (x\#xs) \ G) =$
 $\text{num-of-odd-nodes } G + (\text{if } \text{odd}(\text{degree } v \ G) \wedge v \neq v' \text{ then } -2 \text{ else } 0)$.
thus *?thesis* .
qed
moreover have $\text{odd } (\text{degree } x1 \ (\text{rem-unPath } xs \ G)) \implies$
 $\text{odd}(\text{degree } x3 \ (\text{rem-unPath } xs \ G)) \implies ?\text{case}$
proof –
assume *parity-x1-x3*: $\text{odd } (\text{degree } x1 \ (\text{rem-unPath } xs \ G))$
 $\text{odd } (\text{degree } x3 \ (\text{rem-unPath } xs \ G))$
have $\text{num-of-odd-nodes } (\text{rem-unPath } (x\#xs) \ G) = \text{num-of-odd-nodes}$
 $(\text{del-unEdge } x1 \ x2 \ x3 \ (\text{rem-unPath } xs \ G))$
by (*metis rem-unPath.simps(2) rem-unPath-com x*)
also have $\dots = \text{num-of-odd-nodes } (\text{rem-unPath } xs \ G) - (2::\text{nat})$
using *del-UnEdge-odd-odd*
by (*metis add-implies-diff fin-edges fin-nodes parity-x1-x3 valid-rem-xs x-in*)

also have $\dots = \text{num-of-odd-nodes } G - (2::\text{nat})$
proof –
have $\text{odd}(\text{degree } x3 \ G) \wedge x3 \neq v' \longleftrightarrow \text{even } (\text{degree } x3 \ (\text{rem-unPath } xs \ G))$
using *Cons.prem assms*
by (*metis is-trail.simps(2) parity-x1-x3(2) rem-UnPath-parity-v x*)
hence $\neg(\text{odd}(\text{degree } x3 \ G) \wedge x3 \neq v')$ **by** (*metis parity-x1-x3(2)*)
have $\text{num-of-odd-nodes } (\text{rem-unPath } xs \ G) =$
 $\text{num-of-odd-nodes } G + (\text{if } \text{odd}(\text{degree } x3 \ G) \wedge x3 \neq v' \text{ then } -2 \text{ else } 0)$
by (*metis Cons.hyps Cons.prem(3) assms(1) assms(2)*)
 $\text{is-trail.simps(2) parity-assms } x$
thus *?thesis*
using $\langle \neg (\text{odd } (\text{degree } x3 \ G) \wedge x3 \neq v') \rangle$ **by** *auto*
qed
also have $\dots = \text{num-of-odd-nodes } G + (\text{if } \text{odd}(\text{degree } v \ G) \wedge v \neq v' \text{ then } -2 \text{ else } 0)$
0)
proof –
have $x1 \neq x3$ **by** (*metis valid-rem-xs valid-unMultigraph.no-id x-in*)
moreover hence $x1 \neq v'$
using *Cons.assms*
by (*metis is-trail.simps(2) parity-x1-x3(1) rem-UnPath-parity-v' x*)

ultimately have $x1 \notin \{x3, v'\}$ **by** *auto*
hence $\text{odd}(\text{degree } x1 \ G)$
using *Cons.premis(3) assms(1) assms(2) parity-x1-x3(1)*
by *(metis (full-types) is-trail.simps(2) rem-UnPath-parity-others x)*
hence $\text{odd}(\text{degree } x1 \ G) \wedge x1 \neq v'$ **using** $\langle x1 \neq v' \rangle$ **by** *auto*
hence $\text{odd}(\text{degree } v \ G) \wedge v \neq v'$ **by** *(metis Cons.premis(3) is-trail.simps(2))*
x)
hence $v \in \text{odd-nodes-set } G$
using *Cons.premis(3) E-validD(1) x unfolding odd-nodes-set-def*
by *auto*
moreover have $v' \in \text{odd-nodes-set } G$
using *is-path-memb[OF is-trail-intro[OF assms(3)]] parity-assms*
unfolding *odd-nodes-set-def*
by *auto*
ultimately have $\{v, v'\} \subseteq \text{odd-nodes-set } G$ **by** *auto*
moreover have $v \neq v'$ **by** *(metis (odd (degree v G) $\wedge v \neq v'$))*
hence $\text{card}\{v, v'\} = 2$ **by** *auto*
moreover have $\text{finite}(\text{odd-nodes-set } G)$
using $\langle \text{finite } V \rangle$ **unfolding** *odd-nodes-set-def*
by *auto*
ultimately have $\text{num-of-odd-nodes } G \geq 2$ **by** *(metis card-mono num-of-odd-nodes-def)*

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

lemma *(in valid-unMultigraph) rem-UnPath-cycle:*
assumes *finite E finite V is-trail v ps v' v=v'*
shows $\text{num-of-odd-nodes}(\text{rem-unPath } ps \ G) = \text{num-of-odd-nodes } G$ **(is ?L=?R)**
proof *(cases even(degree v' G))*
case *True*
hence $?L = \text{num-of-odd-nodes } G + (\text{if } \text{even}(\text{degree } v \ G) \wedge v \neq v' \text{ then } 2 \text{ else } 0)$
by *(metis assms(1) assms(2) assms(3) rem-UnPath-even)*
with *assms* **show** *?thesis* **by** *auto*
next
case *False*
hence $?L = \text{num-of-odd-nodes } G + (\text{if } \text{odd}(\text{degree } v \ G) \wedge v \neq v' \text{ then } -2 \text{ else } 0)$
by *(metis assms(1) assms(2) assms(3) rem-UnPath-odd)*
thus *?thesis* **using** $\langle v = v' \rangle$ **by** *auto*
qed

3 Connectivity

definition (in *valid-unMultigraph*) *connected*::bool **where**
connected $\equiv \forall v \in V. \forall v' \in V. v \neq v' \longrightarrow (\exists ps. \text{is-path } v \text{ } ps \text{ } v')$

lemma (in *valid-unMultigraph*) *connected* $\implies \forall v \in V. \forall v' \in V. v \neq v' \longrightarrow (\exists ps. \text{is-trail } v \text{ } ps \text{ } v')$

proof (*rule,rule,rule*)

fix *v v'*

assume $v \in V \ v' \in V \ v \neq v'$

assume *connected*

obtain *ps where is-path v ps v'* **by** (*metis* $\langle \text{connected} \rangle \langle v \in V \rangle \langle v' \in V \rangle \langle v \neq v' \rangle$
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.prem(2) is-path.simps(2) x*)

moreover have $\bigwedge ps'. \text{is-trail } x3 \text{ } ps' \text{ } v' \implies \text{thesis}$

proof –

fix *ps'*

assume *is-trail x3 ps' v'*

hence $(x1, x2, x3) \notin \text{set } ps' \wedge (x3, x2, x1) \notin \text{set } ps' \implies \text{is-trail } v \text{ } (x \# ps') \text{ } v'$

by (*metis Cons.prem(2) is-trail.simps(2) is-path.simps(2) x*)

moreover have $(x1, x2, x3) \in \text{set } ps' \implies \exists ps1. \text{is-trail } v \text{ } ps1 \text{ } v'$

proof –

assume $(x1, x2, x3) \in \text{set } ps'$

then obtain *ps1 ps2 where ps'=ps1@(x1,x2,x3)#ps2* **by** (*metis*
split-list)

hence *is-trail v (x#ps2) v'*

using $\langle \text{is-trail } x3 \text{ } ps' \text{ } v' \rangle x$

by (*metis Cons.prem(2) is-trail.simps(2)*)

is-trail-split is-path.simps(2))

thus ?*thesis* **by** *rule*

qed

moreover have $(x3, x2, x1) \in \text{set } ps' \implies \exists ps1. \text{is-trail } v \text{ } ps1 \text{ } v'$

proof –

assume $(x3, x2, x1) \in \text{set } ps'$

then obtain *ps1 ps2 where ps'=ps1@(x3,x2,x1)#ps2* **by** (*metis*
split-list)

hence *is-trail v ps2 v'*

using $\langle \text{is-trail } x3 \text{ } ps' \text{ } v' \rangle x$

by (*metis Cons.prem(2) is-trail.simps(2)*)

is-trail-split is-path.simps(2))

thus ?*thesis* **by** *rule*

qed

ultimately show *thesis* **using** *Cons* **by** *auto*
qed
ultimately show *?case* **using** *Cons* **by** *auto*
qed
thus $\exists ps. is_trail\ v\ ps\ v'$ **by** *rule*
qed

lemma (in *valid-unMultigraph*) *no-rep-length*: $is_trail\ v\ ps\ v' \implies length\ ps = card(set\ ps)$
by (*induct ps arbitrary:v, auto*)

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

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

definition (in *valid-unMultigraph*) *exist-path-length*:: $'v \Rightarrow nat \Rightarrow bool$ **where**
 $exist_path_length\ v\ l \equiv \exists v'\ ps. is_trail\ v'\ ps\ v \wedge length\ ps = l$

lemma (in *valid-unMultigraph*) *longest-path*:
assumes *finite E n ∈ V*
shows $\exists v. \exists max_path. is_trail\ v\ max_path\ n \wedge$
 $(\forall v'. \forall e \in E. \neg is_trail\ v'\ (e \# max_path)\ n)$
proof (*rule ccontr*)
assume *contro*: $\neg (\exists v\ max_path. is_trail\ v\ max_path\ n$
 $\wedge (\forall v'. \forall e \in E. \neg is_trail\ v'\ (e \# max_path)\ n))$
hence *induct*: $(\forall v\ max_path. is_trail\ v\ max_path\ n$
 $\longrightarrow (\exists v'. \exists e \in E. is_trail\ v'\ (e \# max_path)\ n))$ **by** *auto*
have $is_trail\ n\ []\ n$ **using** $\langle n \in V \rangle$ **by** *auto*
hence $exist_path_length\ n\ 0$ **unfolding** *exist-path-length-def* **by** *auto*
moreover **have** $\forall y. exist_path_length\ n\ y \longrightarrow y \leq card\ E$
using *trail-bound[OF <finite E>]* **unfolding** *exist-path-length-def*
by *auto*
hence *bound*: $\forall y. exist_path_length\ n\ y \longrightarrow y \leq card\ E$ **by** *auto*

ultimately have $\text{exist-path-length } n \text{ (GREATEST } x. \text{ exist-path-length } n \text{ } x)$
using GreatestI-nat **by** auto
then obtain $v \text{ max-path where}$
 $\text{max-path:is-trail } v \text{ max-path } n \text{ length max-path} = (\text{GREATEST } x. \text{ exist-path-length } n \text{ } x)$
by $(\text{metis exist-path-length-def})$
hence $\exists v' e. \text{ is-trail } v' (e \# \text{max-path}) \text{ } n$ **using** induct **by** metis
hence $\text{exist-path-length } n \text{ (length max-path } + 1)$
by $(\text{metis One-nat-def exist-path-length-def list.size(4)})$
hence $\text{length max-path } + 1 \leq (\text{GREATEST } x. \text{ exist-path-length } n \text{ } x)$
by $(\text{metis Greatest-le-nat bound})$
hence $\text{length max-path } + 1 \leq \text{length max-path}$ **using** max-path **by** auto
thus False **by** auto
qed

lemma $\text{even-card}'$:
assumes $\text{even}(\text{card } A) \text{ } x \in A$
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** $(\text{metis all-not-in-conv assms(2) insertI2 mk-disjoint-insert})$
hence $\text{card}(A) = 1$ **by** auto
thus False **using** $\langle \text{even}(\text{card } A) \rangle$ **by** auto
qed

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

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

ps) **by** *auto*
hence *is-trail* $v0$ $((v0,w,v')\#ps)$ v
by (*metis* (*opaque-lifting*, *mono-tags*) *Un-iff* *assms*(2) *corres* *in-set-rev-path* *is-trail.simps*(2))
thus *?thesis* **by** *metis*
qed

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

fun *replace-by-UnPath*:: $('v,'w)$ *path* $\Rightarrow 'v \times 'w \times 'v \Rightarrow ('v,'w)$ *path* $\Rightarrow ('v,'w)$ *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)

lemma (in *valid-unMultigraph*) *del-unEdge-connectivity*:

assumes *connected* $\exists ps.$ *valid-graph.is-path* (*del-unEdge* v e v' G) v ps v'
shows *valid-unMultigraph.connected* (*del-unEdge* v e v' G)

proof –

have *valid-unMulti:valid-unMultigraph* (*del-unEdge* v e v' G)
using *valid-unMultigraph-axioms* **by** *simp*
have *valid-graph: valid-graph* (*del-unEdge* v e v' G)
using *valid-graph-axioms* *del-undirected* **by** (*metis* *delete-edge-valid*)
obtain *ex-path* **where** *ex-path:valid-graph.is-path* (*del-unEdge* v e v' G) v *ex-path* v'

by (*metis* *assms*(2))

show *?thesis* **unfolding** *valid-unMultigraph.connected-def*[*OF* *valid-unMulti*]

proof (*rule,rule,rule*)

fix n n'

assume $n : n \in nodes$ (*del-unEdge* v e v' G)

assume $n' : n' \in nodes$ (*del-unEdge* v e v' G)

assume $n \neq n'$

obtain ps **where** $ps:is-path$ n ps n'

by (*metis* $\langle n \neq n' \rangle$ n n' $\langle connected \rangle$ *connected-def* *del-UnEdge-node*)

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

n (*replace-by-UnPath* ps (v,e,v') *ex-path*) n'

proof (*induct* ps *arbitrary:n*)

case *Nil*

thus *?case* **by** (*metis* *is-path.simps*(1) n' *replace-by-UnPath.simps*(1))

valid-graph

valid-graph.is-path-simps(1))

next

case (*Cons* x xs)

obtain $x1$ $x2$ $x3$ **where** $x:x=(x1,x2,x3)$ **by** (*metis* *prod-cases3*)

have $x=(v,e,v')$ \implies *?case*

proof –

assume $x=(v,e,v')$

hence *valid-graph.is-path* (*del-unEdge* v e v' G)
 n (*replace-by-UnPath* ($x\#xs$) (v,e,v') *ex-path*) n'
 $=$ *valid-graph.is-path* (*del-unEdge* v e v' G)
 n (*ex-path*@(*replace-by-UnPath* xs (v,e,v') *ex-path*)) n'
by (*metis replace-by-UnPath.simps(2)*)
also have ...=*True*
by (*metis Cons.hyps Cons.premis* $\langle x = (v, e, v') \rangle$ *ex-path is-path.simps(2)*)
valid-graph
valid-graph.is-path-split)
finally show ?*thesis* **by** *simp*
qed
moreover have $x=(v',e,v) \implies ?case$
proof –
assume $x=(v',e,v)$
hence *valid-graph.is-path* (*del-unEdge* v e v' G)
 n (*replace-by-UnPath* ($x\#xs$) (v,e,v') *ex-path*) n'
 $=$ *valid-graph.is-path* (*del-unEdge* v e v' G)
 n ((*rev-path ex-path*)@(*replace-by-UnPath* xs (v,e,v') *ex-path*)) n'
by (*metis Cons.premis is-path.simps(2) no-id replace-by-UnPath.simps(2)*)
also have ...=*True*
by (*metis Cons.hyps Cons.premis* $\langle x = (v', e, v) \rangle$ *is-path.simps(2)*)
ex-path valid-graph
valid-graph.is-path-split valid-unMulti valid-unMultigraph.is-path-rev)
finally show ?*thesis* **by** *simp*
qed
moreover have $x\neq(v,e,v') \wedge x\neq(v',e,v) \implies ?case$
by (*metis Cons.hyps Cons.premis del-UnEdge-frame is-path.simps(2)*)
replace-by-UnPath.simps(2)
valid-graph valid-graph.is-path.simps(2) x)
ultimately show ?*case* **by** *auto*
qed
thus $\exists ps.$ *valid-graph.is-path* (*del-unEdge* v e v' G) n ps n' **by** *auto*
qed
qed

lemma (in *valid-unMultigraph*) *path-between-odds*:
assumes *odd*(*degree* v G) *odd*(*degree* v' G) *finite* E $v \neq v'$ *num-of-odd-nodes* $G=2$
shows $\exists ps.$ *is-trail* v ps v'
proof –
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) imageI subsetD*)
hence *degree* v $G=0$ **unfolding** *degree-def* **using** $\langle \text{finite } E \rangle$
by *force*
thus *False* **using** $\langle \text{odd}(\text{degree } v \ G) \rangle$ **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) imageI subsetD)
hence  $\text{degree } v' G = 0$  unfolding degree-def using ⟨finite E⟩
by force
thus False using ⟨odd(degrees v' G)⟩ by auto
qed
then obtain max-path v0 where max-path:
  is-trail v0 max-path v'
  ( $\forall n. \forall w \in E. \neg \text{is-trail } n (w \# \text{max-path}) v'$ )
using longest-path[of v'] by (metis assms(3))
have even(degrees v0 G)  $\implies v0 = v' \implies v0 = v$ 
by (metis assms(2))
moreover have even(degrees v0 G)  $\implies v0 \neq v' \implies v0 = v$ 
proof –
  assume even(degrees v0 G)  $v0 \neq v'$ 
  hence  $\exists w v1. \text{is-trail } v1 (w \# \text{max-path}) v'$ 
  by (metis assms(3) extend-distinct-path max-path(1))
  thus ?thesis by (metis (full-types) is-trail.simps(2) max-path(2) prod.exhaust)
qed
moreover have odd(degrees v0 G)  $\implies v0 = v' \implies v0 = v$ 
proof –
  assume odd(degrees v0 G)  $v0 = v'$ 
  hence  $\exists w v1. \text{is-trail } v1 (w \# \text{max-path}) v'$ 
  by (metis assms(3) extend-distinct-path max-path(1))
  thus ?thesis by (metis (full-types) List.set-simps(2) insert-subset max-path(2)
path-in-edges)
qed
moreover have odd(degrees v0 G)  $\implies v0 \neq v' \implies v0 = v$ 
proof (rule ccontr)
  assume  $v0 \neq v$  odd(degrees v0 G)  $v0 \neq v'$ 
  moreover have  $v \in \text{odd-nodes-set } G$ 
  using ⟨ $v \in V$ ⟩ ⟨odd(degrees v G)⟩ unfolding odd-nodes-set-def
  by auto
  moreover have  $v' \in \text{odd-nodes-set } G$ 
  using ⟨ $v' \in V$ ⟩ ⟨odd(degrees v' 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)
  using assms(5) card-eq-0-iff[of odd-nodes-set G] unfolding num-of-odd-nodes-def

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

```

thus *False* **using** $\langle \text{num-of-odd-nodes } G=2 \rangle$ **unfolding** *num-of-odd-nodes-def*
by *auto*
qed
ultimately have $v0=v$ **by** *auto*
thus *?thesis* **by** (*metis max-path(1)*)
qed

lemma (**in** *valid-unMultigraph*) *del-unEdge-even-connectivity*:
assumes *finite E finite V connected* $\forall n \in V. \text{even}(\text{degree } n \ G)$ $(v, e, v') \in E$
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 (opaque-lifting, 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*: $\forall n \in \text{nodes}(\text{del-unEdge } v \ e \ v' \ G). n \notin \{v, v'\}$
 $\longrightarrow \text{even}(\text{degree } n \ (\text{del-unEdge } v \ e \ v' \ G))$
by (*metis (full-types) assms(1) assms(4) degree-frame del-UnEdge-node*)
have *even (degree v G)* **by** (*metis (full-types) E-validD(1) assms(4) assms(5)*)
moreover have *even (degree v' G)* **by** (*metis (full-types) E-validD(2) assms(4)*
assms(5))
moreover have *num-of-odd-nodes G = 0*
using $\langle \forall n \in V. \text{even}(\text{degree } n \ G) \rangle \langle \text{finite } V \rangle$
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 (degree v (del-unEdge v e v' G))*
using $\langle \text{even}(\text{degree } v \ G) \rangle \text{del-UnEdge-even}'[OF \langle (v, e, v') \in E \rangle \langle \text{finite } E \rangle]$
unfolding *odd-nodes-set-def*
by *auto*
moreover have *odd (degree v' (del-unEdge v e v' G))*
using $\langle \text{even}(\text{degree } v' \ G) \rangle \text{del-UnEdge-even}'[OF \langle (v, e, v') \in E \rangle \langle \text{finite } E \rangle]$
unfolding *odd-nodes-set-def*
by *auto*
moreover have *finite (edges (del-unEdge v e v' G))*
using $\langle \text{finite } E \rangle$ **by** *auto*
moreover have $v \neq v'$ **using** *no-id* $\langle (v, e, v') \in E \rangle$ **by** *auto*
ultimately have $\exists ps. \text{valid-unMultigraph.is-trail}(\text{del-unEdge } v \ e \ v' \ G) \ v \ ps \ v'$
using *valid-unMultigraph.path-between-odds[OF valid-unMulti, of v v']*
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≠[]* \implies *is-path v ps v'* \implies *v'=snd (snd(last ps))*

by (*induct ps arbitrary:v,auto*)

lemma (in *valid-unMultigraph*) *connectivity-split*:

assumes *connected* \neg *valid-unMultigraph.connected (del-unEdge v w v' G)*

$(v,w,v')\in E$

obtains *G1 G2* **where**

nodes G1 = $\{n. \exists ps. \text{valid-graph.is-path } (del-unEdge v w v' G) n ps v\}$

and *edges G1* = $\{(n,e,n'). (n,e,n')\in edges (del-unEdge v w v' G)$

$\wedge n\in nodes G1 \wedge n'\in nodes G1\}$

and *nodes G2* = $\{n. \exists ps. \text{valid-graph.is-path } (del-unEdge v w v' G) n ps v'\}$

and *edges G2* = $\{(n,e,n'). (n,e,n')\in edges (del-unEdge v w v' G)$

$\wedge n\in nodes G2 \wedge n'\in nodes G2\}$

and *edges G1* \cup *edges G2* = *edges (del-unEdge v w v' G)*

and *edges G1* \cap *edges G2* = $\{\}$

and *nodes G1* \cup *nodes G2* = *nodes (del-unEdge v w v' G)*

and *nodes G1* \cap *nodes G2* = $\{\}$

and *valid-unMultigraph G1*

and *valid-unMultigraph G2*

and *valid-unMultigraph.connected G1*

and *valid-unMultigraph.connected G2*

proof –

have *valid0:valid-graph (del-unEdge v w v' G)* **using** *valid-graph-axioms*

by (*metis del-undirected delete-edge-valid*)

have *valid0':valid-unMultigraph (del-unEdge v w v' G)* **using** *valid-unMultigraph-axioms*

by (*metis del-unEdge-valid*)

obtain *G1-nodes* **where** *G1-nodes:G1-nodes* =

$\{n. \exists ps. \text{valid-graph.is-path } (del-unEdge v w v' G) n ps v\}$

by *metis*

then obtain *G1* **where** *G1:G1* =

$(nodes=G1-nodes, edges=\{(n,e,n'). (n,e,n')\in edges (del-unEdge v w v' G)$

$\wedge n\in G1-nodes \wedge n'\in G1-nodes\})$

by *metis*

obtain *G2-nodes* **where** *G2-nodes:G2-nodes* =

$\{n. \exists ps. \text{valid-graph.is-path } (del-unEdge v w v' G) n ps v'\}$

by *metis*

then obtain *G2* **where** *G2:G2* =

$(nodes=G2-nodes, edges=\{(n,e,n'). (n,e,n')\in edges (del-unEdge v w v' G)$

$\wedge n\in G2-nodes \wedge n'\in G2-nodes\})$

by *metis*

have *valid-G1:valid-unMultigraph G1*

using *G1 valid-unMultigraph.corres[OF valid0'] valid-unMultigraph.no-id[OF valid0']*

by (*unfold-locales,auto*)

hence $\text{valid-}G1'$: $\text{valid-graph } G1$ **using** $\text{valid-unMultigraph-def}$ **by** *auto*
have $\text{valid-}G2$: $\text{valid-unMultigraph } G2$
using $G2$ $\text{valid-unMultigraph.corres}$ [*OF valid0'*] $\text{valid-unMultigraph.no-id}$ [*OF valid0'*]
by (*unfold-locales,auto*)
hence $\text{valid-}G2'$: $\text{valid-graph } G2$ **using** $\text{valid-unMultigraph-def}$ **by** *auto*
have $\text{nodes } G1 = \{n. \exists ps. \text{valid-graph.is-path } (\text{del-unEdge } v \ w \ v' \ G) \ n \ ps \ v\}$
using $G1\text{-nodes } G1$ **by** *auto*
moreover **have** $\text{edges } G1 = \{(n,e,n'). (n,e,n') \in \text{edges } (\text{del-unEdge } v \ w \ v' \ G) \wedge n \in \text{nodes } G1 \wedge n' \in \text{nodes } G1\}$
using $G1\text{-nodes } G1$ **by** *auto*
moreover **have** $\text{nodes } G2 = \{n. \exists ps. \text{valid-graph.is-path } (\text{del-unEdge } v \ w \ v' \ G) \ n \ ps \ v'\}$
using $G2\text{-nodes } G2$ **by** *auto*
moreover **have** $\text{edges } G2 = \{(n,e,n'). (n,e,n') \in \text{edges } (\text{del-unEdge } v \ w \ v' \ G) \wedge n \in \text{nodes } G2 \wedge n' \in \text{nodes } G2\}$
using $G2\text{-nodes } G2$ **by** *auto*
moreover **have** $\text{nodes } G1 \cup \text{nodes } G2 = \text{nodes } (\text{del-unEdge } v \ w \ v' \ G)$
proof (*rule ccontr*)
assume $\text{nodes } G1 \cup \text{nodes } G2 \neq \text{nodes } (\text{del-unEdge } v \ w \ v' \ G)$
moreover **have** $\text{nodes } G1 \subseteq \text{nodes } (\text{del-unEdge } v \ w \ v' \ G)$
using $\text{valid-graph.is-path-memb}$ [*OF valid0'*] $G1$ $G1\text{-nodes}$ **by** *auto*
moreover **have** $\text{nodes } G2 \subseteq \text{nodes } (\text{del-unEdge } v \ w \ v' \ G)$
using $\text{valid-graph.is-path-memb}$ [*OF valid0'*] $G2$ $G2\text{-nodes}$ **by** *auto*
ultimately obtain n **where** n :
 $n \in \text{nodes } (\text{del-unEdge } v \ w \ v' \ G) \ n \notin \text{nodes } G1 \ n \notin \text{nodes } G2$
by *auto*
hence $n\text{-neg-}v : \neg(\exists ps. \text{valid-graph.is-path } (\text{del-unEdge } v \ w \ v' \ G) \ n \ ps \ v)$ **and**
 $n\text{-neg-}v' : \neg(\exists ps. \text{valid-graph.is-path } (\text{del-unEdge } v \ w \ v' \ G) \ n \ ps \ v')$
using $G1$ $G1\text{-nodes}$ $G2$ $G2\text{-nodes}$ **by** *auto*
hence $n \neq v$ **by** (*metis n(1) valid0' valid-graph.is-path-simps(1)*)
then obtain nvs **where** nvs : $\text{is-path } n \ nvs \ v$ **using** $\langle \text{connected} \rangle$
by (*metis E-validD(1) assms(3) connected-def del-UnEdge-node n(1)*)
then obtain nvs' **where** $nvs' = \text{takeWhile } (\lambda x. x \neq (v,w,v') \wedge x \neq (v',w,v))$
 nvs **by** *auto*
moreover **have** $nvs - nvs' : nvs = nvs' @ \text{dropWhile } (\lambda x. x \neq (v,w,v') \wedge x \neq (v',w,v))$
 nvs
using nvs' $\text{takeWhile-dropWhile-id}$ **by** *auto*
ultimately obtain n' **where** $\text{is-path-}nvs'$: $\text{is-path } n \ nvs' \ n'$
and $\text{is-path } n'$ ($\text{dropWhile } (\lambda x. x \neq (v,w,v') \wedge x \neq (v',w,v)) \ nvs$) v
using nvs is-path-split [*of n nvs' dropWhile } (\lambda x. x \neq (v,w,v') \wedge x \neq (v',w,v)) \ nvs]
by *auto*
have $n' = v \vee n' = v'$
proof (*cases dropWhile } (\lambda x. x \neq (v,w,v') \wedge x \neq (v',w,v)) \ nvs*)
case Nil
hence $nvs = nvs'$ **using** $nvs - nvs'$ **by** (*metis append-Nil2*)
hence $n' = v$ **using** nvs $\text{is-path-}nvs'$ path-end **by** (*metis (mono-tags) is-path.simps(1)*)
thus $?thesis$ **by** *auto**

```

next
  case (Cons x xs)
  hence dropWhile (λx. x≠(v,w,v')∧x≠(v',w,v)) nvs≠[] by auto
  hence hd (dropWhile (λx. x≠(v,w,v')∧x≠(v',w,v)) nvs)=(v,w,v')
    ∨ hd (dropWhile (λx. x≠(v,w,v')∧x≠(v',w,v)) nvs)=(v',w,v)
    by (metis (lifting, full-types) hd-dropWhile)
  hence x=(v,w,v')∨x=(v',w,v) using Cons by auto
  thus ?thesis
    using ⟨is-path n' (dropWhile (λx. x ≠ (v, w, v') ∧ x ≠ (v', w, v)) nvs)
v⟩
    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' nvs'
  proof (induct nvs' arbitrary:n nvs)
    case Nil
  thus ?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 xs n' using Cons by auto
  moreover have xs = takeWhile (λx. x ≠ (v, w, v') ∧ x ≠ (v', w, v)) (tl
nvs)
    using ⟨x ≠ xs = takeWhile (λx. x ≠ (v, w, v') ∧ x ≠ (v', w, v)) nvs⟩
  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≠(v,w,v') ∧ x≠(v',w,v)
    using Cons(3) set-takeWhileD[of x (λx. x ≠ (v, w, v') ∧ x ≠ (v', w, v))]
nvs]
    by (metis List.set.simps(2) insertI1)
  hence x∈edges (del-unEdge v w v' G)
    by (metis Cons.prem(1) del-UnEdge-frame is-path.simps(2) x)
  ultimately show ?case using x
  by (metis Cons.prem(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 ∩ nodes G2={ }
  proof (rule ccontr)
  assume nodes G1 ∩ nodes G2 ≠ { }
  then obtain n where n:n∈nodes G1 n∈nodes G2 by auto
  then obtain nvs nv's where
    nvs : valid-graph.is-path (del-unEdge v w v' G) n nvs v and
    nv's : valid-graph.is-path (del-unEdge v w v' G) n nv's v'
  using G1 G2 G1-nodes G2-nodes by auto
  hence valid-graph.is-path (del-unEdge v w v' G) v ((rev-path nvs)@nv's) 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' G)
    by (metis assms(1) del-unEdge-connectivity)
    thus False by (metis assms(2))
qed
moreover have edges G1  $\cup$  edges G2 = edges (del-unEdge v w v' G)
proof (rule ccontr)
  assume edges G1  $\cup$  edges G2  $\neq$  edges (del-unEdge v w v' G)
  moreover have edges G1  $\subseteq$  edges (del-unEdge v w v' G) using G1 by auto
  moreover have edges G2  $\subseteq$  edges (del-unEdge v w v' G) using G2 by auto
  ultimately obtain n e n' where
    nen':
    (n,e,n') $\in$ edges (del-unEdge v w v' G)
    (n,e,n') $\notin$ edges G1 (n,e,n') $\notin$ edges G2
  by auto
  moreover have n $\in$ nodes (del-unEdge v w v' G)
  by (metis nen'(1) valid0 valid-graph.E-validD(1))
  moreover have n' $\in$ nodes (del-unEdge v w v' G)
  by (metis nen'(1) valid0 valid-graph.E-validD(2))
  ultimately have (n $\in$ nodes G1  $\wedge$  n' $\in$ nodes G2) $\vee$ (n $\in$ nodes G2 $\wedge$ n' $\in$ nodes
G1)
  using G1 G2  $\langle$ nodes G1  $\cup$  nodes G2=nodes (del-unEdge v w v' G) $\rangle$  by auto
  moreover have n $\in$ nodes G1  $\implies$  n' $\in$ nodes G2  $\implies$  False
  proof -
    assume n $\in$ nodes G1 n' $\in$ nodes G2
    then obtain nvs nv's where
      nvs : valid-graph.is-path (del-unEdge v w v' G) n nvs v and
      nv's : valid-graph.is-path (del-unEdge v w v' G) n' nv's v'
    using G1 G2 G1-nodes G2-nodes by auto
    hence valid-graph.is-path (del-unEdge v w 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]
       $\langle$ (n,e,n') $\in$ edges (del-unEdge v w v' G) $\rangle$ 
    by auto
    hence valid-unMultigraph.connected (del-unEdge v w v' G)
    by (metis assms(1) del-unEdge-connectivity)
    thus False by (metis assms(2))
  qed
  moreover have n $\in$ nodes G2  $\implies$  n' $\in$ nodes G1  $\implies$  False
  proof -
    assume n' $\in$ nodes G1 n $\in$ nodes G2
    then obtain n'vs nvs where
      n'vs : valid-graph.is-path (del-unEdge v w v' G) n' n'vs v and
      nvs : valid-graph.is-path (del-unEdge v w v' G) n nvs v'
    using G1 G2 G1-nodes G2-nodes by auto
    moreover have (n',e,n) $\in$ edges (del-unEdge v w v' G)

```



```

    by (metis nen'(1) valid0' valid-unMultigraph.corres)
    ultimately have valid-graph.is-path (del-unEdge v w v' G) v
      ((rev-path n'vs)@(n',e,n)#nvs) 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' G)
      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' G) n ps v
      using G1 G1-nodes n by auto
    hence ps:valid-graph.is-path G1 n ps v
    proof (induct ps arbitrary:n)
      case Nil
      moreover have v∈nodes G1 using G1 G1-nodes valid0
        by (metis (lifting, no-types) calculation mem-Collect-eq select-convs(1)
          valid-graph.is-path.simps(1))
      ultimately show ?case
        by (metis valid0 valid-G1 valid-unMultigraph.is-trail.simps(1)
          valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)
    next
      case (Cons x xs)
      obtain x1 x2 x3 where x:x=(x1,x2,x3) by (metis prod-cases3)
      have x1∈nodes G1 using G1 G1-nodes Cons.prem 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.prem valid0 valid-graph.is-path.simps(2) x)
      ultimately have (x1,x2,x3)∈edges G1
        using G1 G2 ⟨nodes G1 ∩ nodes G2={ }⟩ ⟨edges G1 ∪ edges G2=edges
(del-unEdge v w v' G)⟩
      by (metis (full-types) IntI Un-iff be-empty valid-G2' valid-graph.E-validD(1)
)

```

moreover have *valid-graph.is-path* (*del-unEdge* *v w v' G*) *x3 xs v*
by (*metis Cons.premis 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.premis 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' G*) *n' ps' v*
using *G1 G1-nodes n'* **by** *auto*
hence *ps':valid-graph.is-path* *G1 n' ps' v*
proof (*induct ps' arbitrary:n'*)
case *Nil*
moreover have *v∈nodes G1* **using** *G1 G1-nodes valid0*
by (*metis (lifting, no-types) calculation mem-Collect-eq select-convs(1)*
valid-graph.is-path.simps(1))
ultimately show *?case*
by (*metis valid0 valid-G1 valid-unMultigraph.is-trail.simps(1)*
valid-graph.is-path.simps(1) valid-unMultigraph.is-trail-intro)
next
case (*Cons x xs*)
obtain *x1 x2 x3* **where** *x:x=(x1,x2,x3)* **by** (*metis prod-cases3*)
have *x1∈nodes G1* **using** *G1 G1-nodes Cons.premis 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.premis valid0 valid-graph.is-path.simps(2)* *x*)
ultimately have *(x1,x2,x3)∈edges G1*
using *G1 G2 ⟨nodes G1 ∩ nodes G2=⟩*
⟨edges G1 ∪ edges G2=edges (del-unEdge v w v' G)⟩
by (*metis (full-types) IntI Un-iff bex-empty valid-G2' valid-graph.E-validD(1)*)
moreover have *valid-graph.is-path* (*del-unEdge* *v w v' G*) *x3 xs v*
by (*metis Cons.premis 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.premis valid0 valid-graph.is-path.simps(2)*)
x) **ultimately show** *?case* **using** *x valid-G1'* **by** (*metis valid-graph.is-path.simps(2)*)

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 $\exists ps.$ *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 ∈ nodes G2
assume n' : n' ∈ nodes G2
assume n ≠ n'
obtain ps where valid-graph.is-path (del-unEdge v w v' G) n ps v'
  using G2 G2-nodes n by auto
hence ps : valid-graph.is-path G2 n ps v'
proof (induct ps arbitrary : n)
  case Nil
  moreover have v' ∈ nodes G2 using G2 G2-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 xs)
  obtain x1 x2 x3 where x : x = (x1, x2, x3) by (metis prod-cases3)
  have x1 ∈ nodes G2 using G2 G2-nodes Cons.prem1 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.prem1 valid0 valid-graph.is-path.simps(2) x)
  ultimately have (x1, x2, x3) ∈ edges G2
    using ⟨nodes G1 ∩ nodes G2 = {}⟩ ⟨edges G1 ∪ edges G2 = edges
      (del-unEdge v w v' G)⟩
    by (metis IntI Un-iff assms(1) be-empty connected-def del-UnEdge-node
      valid0 valid0')
  valid-G1' valid-graph.E-validD(1) valid-graph.E-validD(2) valid-unMultigraph.no-id
  moreover have valid-graph.is-path (del-unEdge v w v' G) x3 xs v'
    by (metis Cons.prem1 valid0 valid-graph.is-path.simps(2) x)
  hence valid-graph.is-path G2 x3 xs v' using Cons.hyps by auto
  moreover have x1 = n by (metis Cons.prem1 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 valid-graph.is-path (del-unEdge v w v' G) n' ps' v'
  using G2 G2-nodes n' by auto
hence ps' : valid-graph.is-path G2 n' ps' v'
proof (induct ps' arbitrary : n')
  case Nil
  moreover have v' ∈ nodes G2 using G2 G2-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 xs)

```

obtain $x1\ x2\ x3$ **where** $x:x=(x1,x2,x3)$ **by** (*metis prod-cases3*)
have $x1 \in \text{nodes } G2$ **using** $G2\ G2\text{-nodes}\ \text{Cons.prem}s\ x$
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.prem}s\ \text{valid0}\ \text{valid-graph.is-path.simps}(2)\ x*)
ultimately have $(x1,x2,x3) \in \text{edges } G2$
using $\langle \text{nodes } G1 \cap \text{nodes } G2 = \{\} \rangle\ \langle \text{edges } G1 \cup \text{edges } G2 = \text{edges}$
 $(\text{del-unEdge } v\ w\ v'\ G) \rangle$
by (*metis IntI Un-iff assms(1) bex-empty connected-def del-UnEdge-node*
valid0 valid0'
valid-G1' valid-graph.E-validD(1) valid-graph.E-validD(2) valid-unMultigraph.no-id)
moreover have $\text{valid-graph.is-path } (\text{del-unEdge } v\ w\ v'\ G)\ x3\ xs\ v'$
by (*metis Cons.prem}s\ \text{valid0}\ \text{valid-graph.is-path.simps}(2)\ x*)
hence $\text{valid-graph.is-path } G2\ x3\ xs\ v'$ **using** Cons.hyps **by** *auto*
moreover have $x1 = n'$ **by** (*metis Cons.prem}s\ \text{valid0}\ \text{valid-graph.is-path.simps}(2)*
 x)
ultimately show $?case$ **using** $x\ \text{valid-G2}'$ **by** (*metis valid-graph.is-path.simps(2)*)

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

lemma *sub-graph-degree-frame:*

assumes $\text{valid-graph } G2\ \text{edges } G1 \cup \text{edges } G2 = \text{edges } G\ \text{nodes } G1 \cap \text{nodes}$
 $G2 = \{\}\ n \in \text{nodes } G1$

shows $\text{degree } n\ G = \text{degree } n\ G1$

proof –

have $\{e \in \text{edges } G.\ \text{fst } e = n\} \subseteq \{e \in \text{edges } G1.\ \text{fst } e = n\}$

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 $n \notin \text{nodes } G2$

using $\langle \text{nodes } G1 \cap \text{nodes } G2 = \{\} \rangle\ \langle n \in \text{nodes } G1 \rangle$

by *auto*

hence $e \notin \text{edges } G2$ **using** $\text{valid-graph.E-validD}[OF\ \langle \text{valid-graph } G2 \rangle]\ \langle \text{fst } e = n \rangle$

by (*metis prod.exhaust fst-conv*)

ultimately have $e \in \text{edges } G1$ **using** $\langle \text{edges } G1 \cup \text{edges } G2 = \text{edges } G \rangle$ **by**
auto

thus $e \in \{e \in \text{edges } G1.\ \text{fst } e = n\}$ **using** $\langle \text{fst } e = n \rangle$ **by** *auto*

qed
moreover have $\{e \in \text{edges } G1. \text{fst } e = n\} \subseteq \{e \in \text{edges } G. \text{fst } e = n\}$
by (*metis (lifting) Collect-mono Un-iff assms(2)*)
ultimately show *?thesis unfolding degree-def by auto*
qed

lemma *odd-nodes-no-edge[simp]: finite (nodes g) \implies 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 bool where*
adjacent v v' \equiv $\exists w. (v,w,v') \in E$

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

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

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

lemma (*in valid-unMultigraph*) *adjacent-finite:*
finite E \implies finite {n. adjacent v n}

proof –

assume *finite E*

{ fix *S v*

have *finite S \implies finite {n. $\exists w. (v,w,n) \in S$ }*

proof (*induct S rule: finite-induct*)

case *empty*

thus *?case by auto*

next

case (*insert x F*)

obtain *x1 x2 x3 where x: x=(x1,x2,x3) by (metis prod-cases3)*

have *x1=v \implies ?case*

proof –

assume *x1=v*

hence *{n. $\exists w. (v, w, n) \in \text{insert } x F$ } = \text{insert } x3 \{n. $\exists w. (v, w, n) \in F$ }*

using *x by auto*

thus *?thesis using insert by auto*

qed

moreover have *x1 \neq v \implies ?case*

proof –

assume *x1 \neq v*

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 insert by auto*
 qed
 ultimately show *?case by auto*
 qed }
 note *aux=this*
 show *?thesis using aux[OF <finite E>, of v] unfolding adjacent-def by auto*
 qed

5 Undirected simple graph

locale *valid-unSimpGraph=valid-unMultigraph G for G::('v,'w) graph+*
 assumes *no-multi[simp]: (v,w,u) ∈ edges G ⇒ (v,w',u) ∈ edges G ⇒*
w = w'

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*

next

case *False*

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

moreover have *finite (V × 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} ≠ 0 using False card-eq-0-iff by auto*

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

proof –

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

proof –

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

unfolding adjacent-def by auto

moreover have $\bigwedge x. x \in \{(v1,v2). \text{adjacent } v1 \ v2\} \implies x \in (\lambda(v1,w,v2).$

$(v1,v2))'E$

unfolding adjacent-def by force

ultimately show ?thesis by force

qed

moreover have *inj-on (λ(v1,w,v2). (v1,v2)) E unfolding inj-on-def by auto*

ultimately show ?thesis by (metis card-image)

qed

ultimately show finite E by (metis card.infinite)

qed

lemma *del-unEdge-valid'[simp]: valid-unSimpGraph G ⇒*

$valid-unSimpGraph (del-unEdge v w u G)$
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 $valid-unSimpGraph.no-multi[OF \langle valid-unSimpGraph G \rangle]$
 unfolding $valid-unSimpGraph-axioms-def del-unEdge-def$ **by auto**
 ultimately show $valid-unSimpGraph (del-unEdge v w u G)$ **using** $valid-unSimpGraph-def$
by auto
qed

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

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

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

using $valid-unSimpGraph-axioms$
proof (induct $degree v G$ arbitrary: G)
 case 0
 note $valid3 = \langle valid-unSimpGraph G \rangle$
 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 $(v,w,u) \in edges G$
 unfolding $valid-unMultigraph.adjacent-def[OF valid2]$ **by auto**
 hence $degree v G \neq 0$ **using** $\langle finite (edges G) \rangle$ **unfolding** $degree-def$ **by auto**
 thus $False$ **using** $\langle 0 = degree v G \rangle$ **by auto**
qed

thus ?case **by** (metis 0.hyps card.empty)

next

case (Suc n)
 hence $\{e \in edges G. fst e = v\} \neq \{\}$ **using** $card.empty$ **unfolding** $degree-def$ **by**
force
 then obtain $w u$ where $(v,w,u) \in edges G$ **by auto**
 have $valid: valid-unMultigraph G$ **using** $\langle valid-unSimpGraph G \rangle valid-unSimpGraph-def$

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'* $\langle \text{valid-unSimpGraph } G \rangle$ **by** *auto*
moreover have $n = \text{degree } v$ (*del-unEdge* *v w u G*)
using $\langle \text{Suc } n = \text{degree } v \text{ } G \rangle \langle (v, w, u) \in \text{edges } G \rangle$ *del-edge-undirected-degree-plus*[*of*
G v w u]
by (*metis Suc.prem1 Suc-eq-plus1 diff-Suc-1 valid valid-unMultigraph.corres*)
moreover have *finite* (*edges* (*del-unEdge* *v w u G*))
using $\langle \text{finite } (\text{edges } G) \rangle$ **unfolding** *del-unEdge-def*
by *auto*
ultimately have $\text{degree } v$ (*del-unEdge* *v w u G*)
 $= \text{card } (\text{Collect } (\text{valid-unMultigraph.adjacent } (\text{del-unEdge } v w u G) v))$
using *Suc.hyps* **by** *auto*
moreover have *Suc*($\text{card } (\{n. \text{valid-unMultigraph.adjacent } (\text{del-unEdge } v w u$
G)
 $v n\})) = \text{card } (\{n. \text{valid-unMultigraph.adjacent } G v n\})$
using *valid-unMultigraph.adjacent-def*[*OF valid'*]
proof –
have $\{n. \text{valid-unMultigraph.adjacent } (\text{del-unEdge } v w u G) v n\} \subseteq$
 $\{n. \text{valid-unMultigraph.adjacent } G v n\}$
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 $u \in \{n. \text{valid-unMultigraph.adjacent } G v n\}$
using $\langle (v, w, u) \in \text{edges } G \rangle$ **unfolding** *valid-unMultigraph.adjacent-def*[*OF*
valid'] **by** *auto*
ultimately have $\{n. \text{valid-unMultigraph.adjacent } (\text{del-unEdge } v w u G) v n\}$
 $\cup \{u\}$
 $\subseteq \{n. \text{valid-unMultigraph.adjacent } G v n\}$ **by** *auto*
moreover have $\{n. \text{valid-unMultigraph.adjacent } G v n\} - \{u\}$
 $\subseteq \{n. \text{valid-unMultigraph.adjacent } (\text{del-unEdge } v w u G) v n\}$
using *del-unEdge-def*[*of v w u G*]
unfolding *valid-unMultigraph.adjacent-def*[*OF valid'*]
valid-unMultigraph.adjacent-def[*OF valid'*]
by *auto*
ultimately have $\{n. \text{valid-unMultigraph.adjacent } (\text{del-unEdge } v w u G) v n\}$
 $\cup \{u\}$
 $= \{n. \text{valid-unMultigraph.adjacent } G v n\}$ **by** *auto*
moreover have $u \notin \{n. \text{valid-unMultigraph.adjacent } (\text{del-unEdge } v w u G) v$
n
 $\}$
using *valid-unSimpGraph.del-UnEdge-non-adj*[*OF* $\langle \text{valid-unSimpGraph } G \rangle$
 $\langle (v, w, u) \in \text{edges } G \rangle$]
by *auto*
moreover have *finite* $\{n. \text{valid-unMultigraph.adjacent } G v n\}$
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 = degree v (del-unEdge v w u
 G)›)
 qed
 end

theory *KoenigsbergBridge* imports *MoreGraph*
 begin

6 Definition of Eulerian trails and circuits

definition (in *valid-unMultigraph*) *is-Eulerian-trail*:: 'v ⇒ ('v,'w) path ⇒ 'v ⇒ bool
where
is-Eulerian-trail v ps v' ≡ is-trail v ps v' ∧ edges (rem-unPath ps G) = {}

definition (in *valid-unMultigraph*) *is-Eulerian-circuit*:: 'v ⇒ ('v,'w) path ⇒ 'v ⇒ bool
where
is-Eulerian-circuit v ps v' ≡ (v=v') ∧ (is-Eulerian-trail v ps v')

7 Necessary conditions for Eulerian trails and circuits

lemma (in *valid-unMultigraph*) *euclerian-rev*:
is-Eulerian-trail v' (rev-path ps) v = is-Eulerian-trail v ps v'

proof –

have *is-trail v' (rev-path ps) v = is-trail v ps v'*

by (metis *is-trail-rev*)

moreover have *edges (rem-unPath (rev-path ps) G) = edges (rem-unPath ps G)*

by (metis *rem-unPath-graph*)

ultimately show ?thesis **unfolding** *is-Eulerian-trail-def* **by** *auto*

qed

theorem (in *valid-unMultigraph*) *euclerian-cycle-ex*:

assumes *is-Eulerian-circuit v ps v' finite V finite E*

shows $\forall v \in V. \text{even } (\text{degree } v \ G)$

proof –

obtain *v ps v' where cycle:is-Eulerian-circuit v ps v' using* *assms* **by** *auto*

hence *edges (rem-unPath ps G) = {}*

unfolding *is-Eulerian-circuit-def is-Eulerian-trail-def*

by *simp*

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

ultimately have *rem-unPath ps G = G (edges:={})* **by** *auto*

hence *num-of-odd-nodes (rem-unPath ps G) = 0* **by** (metis *assms(2) odd-nodes-no-edge*)

moreover have *v = v'*

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

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

```

0)=0
  using odd-nodes by auto
have odd (degree v G)  $\implies$  ?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)  $\implies$  ?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'  $\implies$  ?thesis
  by (metis assms(2) assms(3) euclerian-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  $\equiv$  (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  $\longleftrightarrow$  P a  $\wedge$  P b  $\wedge$  P c  $\wedge$  P d

```

```

definition [simp]: enum-class.enum-ex P  $\longleftrightarrow$  P a  $\vee$  P b  $\vee$  P c  $\vee$  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  $P \longleftrightarrow P \text{ ab1} \wedge P \text{ ab2} \wedge P \text{ ac1} \wedge P \text{ ac2}$ 
 $\wedge P \text{ ad1} \wedge P \text{ bd1}$ 
 $\wedge P \text{ cd1}$ 
definition [simp]:enum-class.enum-ex  $P \longleftrightarrow P \text{ ab1} \vee P \text{ ab2} \vee P \text{ ac1} \vee P \text{ ac2}$ 
 $\vee P \text{ ad1} \vee P \text{ bd1}$ 
 $\vee P \text{ cd1}$ 
instance proof qed (auto,(case-tac x,auto)+)
end

interpretation kon-graph: valid-unMultigraph kon-graph
proof (unfold-locales)
  show fst ‘ edges kon-graph  $\subseteq$  nodes kon-graph by eval
next
  show snd ‘ snd ‘ edges kon-graph  $\subseteq$  nodes kon-graph by eval
next
  have  $\forall v w u'. ((v, w, u') \in \text{edges } \textit{kon-graph}) = ((u', w, v) \in \text{edges } \textit{kon-graph})$ 
  by eval
  thus  $\bigwedge v w u'. ((v, w, u') \in \text{edges } \textit{kon-graph}) = ((u', w, v) \in \text{edges } \textit{kon-graph})$ 
by simp
next
  have  $\forall v w. (v, w, v) \notin \text{edges } \textit{kon-graph}$  by eval
  thus  $\bigwedge v w. (v, w, v) \notin \text{edges } \textit{kon-graph}$  by simp
qed

theorem  $\neg \textit{kon-graph.is-Eulerian-trail } v1 \text{ p } v2$ 
proof
  assume kon-graph.is-Eulerian-trail } v1 \text{ 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:
  ( $\forall v \in \text{nodes } \textit{kon-graph}. \text{even } (\text{degree } v \textit{ kon-graph})$ )  $\vee$  (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  $\neg(\text{num-of-odd-nodes } \textit{kon-graph} =2)$  by eval
  moreover have  $\neg(\forall v \in \text{nodes } \textit{kon-graph}. \text{even } (\text{degree } v \textit{ kon-graph}))$  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 w v1 G*) *v1 ps v2*
 $(v0, w, v1) \in E$
shows *is-Eulerian-trail* *v0 ((v0, w, v1) # ps) v2*
proof –
have *valid:valid-unMultigraph* (*del-unEdge* *v0 w v1 G*)
using *valid-unMultigraph-axioms* **by** *auto*
hence *distinct:valid-unMultigraph.is-trail* (*del-unEdge* *v0 w v1 G*) *v1 ps v2*
using *assms unfolding valid-unMultigraph.is-Eulerian-trail-def* [*OF valid*]
by *auto*
hence $set\ ps \subseteq edges$ (*del-unEdge* *v0 w v1 G*)
using *valid-unMultigraph.path-in-edges* [*OF valid*] **by** *auto*
moreover **have** $(v0, w, v1) \notin edges$ (*del-unEdge* *v0 w v1 G*)
unfolding *del-unEdge-def* **by** *auto*
moreover **have** $(v1, w, v0) \notin edges$ (*del-unEdge* *v0 w v1 G*)
unfolding *del-unEdge-def* **by** *auto*
ultimately **have** $(v0, w, v1) \notin set\ ps$ $(v1, w, v0) \notin set\ ps$ **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 $\langle (v0, w, v1) \in E \rangle$ **by** *auto*
moreover **have** $edges$ (*rem-unPath* *ps* (*del-unEdge* *v0 w v1 G*)) = {}
using *assms unfolding valid-unMultigraph.is-Eulerian-trail-def* [*OF valid*]
by *auto*
hence $edges$ (*rem-unPath* $((v0, w, v1) \# ps)$ *G*) = {}
by (*metis rem-unPath.simps*(2))
ultimately **show** *?thesis* **unfolding** *is-Eulerian-trail-def* **by** *auto*
qed

lemma (in *valid-unMultigraph*) *eulerian-cons'*:
assumes
valid-unMultigraph.is-Eulerian-trail (*del-unEdge* *v2 w v3 G*) *v1 ps v2*
 $(v2, w, v3) \in E$
shows *is-Eulerian-trail* *v1 (ps @ [(v2, w, v3)]) v3*
proof –
have *valid:valid-unMultigraph* (*del-unEdge* *v3 w v2 G*)
using *valid-unMultigraph-axioms del-unEdge-valid* **by** *auto*
have *del-unEdge* *v2 w v3 G* = *del-unEdge* *v3 w v2 G*
by (*metis delete-edge-sym*)
hence *valid-unMultigraph.is-Eulerian-trail* (*del-unEdge* *v3 w v2 G*) *v2*
 $(rev-path\ ps)$ *v1* **using** *assms valid-unMultigraph.eulerian-rev* [*OF valid*]
by *auto*
hence *is-Eulerian-trail* *v3 ((v3, w, v2) # (rev-path ps)) v1*
using *eulerian-cons* **by** (*metis assms*(2) *corres*)
hence *is-Eulerian-trail* *v1 (rev-path((v3, w, v2) # (rev-path ps))) v3*

using *euclerian-rev* **by** *auto*
moreover have $\text{rev-path}((v3,w,v2)\#(\text{rev-path } ps)) = \text{rev-path}(\text{rev-path } ps)\@[v2,w,v3]$
unfolding *rev-path-def* **by** *auto*
hence $\text{rev-path}((v3,w,v2)\#(\text{rev-path } ps))=ps\@[v2,w,v3]$ **by** *auto*
ultimately show *?thesis* **by** *auto*
qed

lemma *eulerian-split*:

assumes $\text{nodes } G1 \cap \text{nodes } G2 = \{\}$ $\text{edges } G1 \cap \text{edges } G2 = \{\}$
 $\text{valid-unMultigraph } G1$ $\text{valid-unMultigraph } G2$
 $\text{valid-unMultigraph.is-Eulerian-trail } G1$ $v1$ $ps1$ $v1'$
 $\text{valid-unMultigraph.is-Eulerian-trail } G2$ $v2$ $ps2$ $v2'$
shows $\text{valid-unMultigraph.is-Eulerian-trail } (\text{nodes}=\text{nodes } G1 \cup \text{nodes } G2,$
 $\text{edges}=\text{edges } G1 \cup \text{edges } G2 \cup \{(v1',w,v2),(v2,w,v1')\})$ $v1$ $(ps1\@[v1',w,v2]\#ps2)$
 $v2'$

proof –

have $\text{valid-graph } G1$ **using** $\langle \text{valid-unMultigraph } G1 \rangle$ $\text{valid-unMultigraph-def}$ **by** *auto*

have $\text{valid-graph } G2$ **using** $\langle \text{valid-unMultigraph } G2 \rangle$ $\text{valid-unMultigraph-def}$ **by** *auto*

obtain G **where** $G:G=(\text{nodes}=\text{nodes } G1 \cup \text{nodes } G2, \text{edges}=\text{edges } G1 \cup \text{edges } G2$

$\cup \{(v1',w,v2),(v2,w,v1')\})$

by *metis*

have $v1' \in \text{nodes } G1$

by $(\text{metis } (\text{full-types}) \langle \text{valid-graph } G1 \rangle \text{assms}(3) \text{assms}(5) \text{valid-graph.is-path-memb}$
 $\text{valid-unMultigraph.is-trail-intro } \text{valid-unMultigraph.is-Eulerian-trail-def})$

moreover have $v2 \in \text{nodes } G2$

by $(\text{metis } (\text{full-types}) \langle \text{valid-graph } G2 \rangle \text{assms}(4) \text{assms}(6) \text{valid-graph.is-path-memb}$
 $\text{valid-unMultigraph.is-trail-intro } \text{valid-unMultigraph.is-Eulerian-trail-def})$

moreover have $\langle ba \in \text{nodes } G1 \rangle$ **if** $\langle (aa, ab, ba) \in \text{edges } G1 \rangle$

for aa ab ba

using *that*

by $(\text{meson } \langle \text{valid-graph } G1 \rangle \text{valid-graph.E-validD}(2))$

ultimately have $\text{valid-unMultigraph } (\text{nodes}=\text{nodes } G1 \cup \text{nodes } G2, \text{edges}=\text{edges } G1 \cup \text{edges } G2 \cup$

$\{(v1',w,v2),(v2,w,v1')\})$

using

$\text{valid-unMultigraph.corres}[OF \langle \text{valid-unMultigraph } G1 \rangle]$

$\text{valid-unMultigraph.no-id}[OF \langle \text{valid-unMultigraph } G1 \rangle]$

$\text{valid-unMultigraph.corres}[OF \langle \text{valid-unMultigraph } G2 \rangle]$

$\text{valid-unMultigraph.no-id}[OF \langle \text{valid-unMultigraph } G2 \rangle]$

$\text{valid-graph.E-validD}[OF \langle \text{valid-graph } G1 \rangle]$

$\text{valid-graph.E-validD}[OF \langle \text{valid-graph } G2 \rangle]$

$\langle \text{nodes } G1 \cap \text{nodes } G2 = \{\} \rangle$

by *unfold-locales auto*

hence $\text{valid}: \text{valid-unMultigraph } G$ **using** G **by** *auto*

hence $\text{valid}': \text{valid-graph } G$ **using** $\text{valid-unMultigraph-def}$ **by** *auto*

moreover have $\text{valid-unMultigraph.is-trail } G$ $v1$ $(ps1\@[v1',w,v2]\#ps2)$ $v2'$

```

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) se-
lect-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* $\langle \text{edges } G1 \cap \text{edges } G2 = \{\} \rangle$ **by auto**
qed

moreover have $(v1', w, v2) \notin \text{edges } G1$
using $\langle v2 \in \text{nodes } G2 \rangle \langle \text{valid-graph } G1 \rangle$
by (*metis Int-iff all-not-in-conv* *assms(1) valid-graph.E-validD(2)*)
hence $(v1', w, v2) \notin \text{set } ps1$
by (*metis (full-types) assms(3) assms(5) subsetD valid-unMultigraph.path-in-edges*
valid-unMultigraph.is-Eulerian-trail-def)
moreover have $(v1', w, v2) \notin \text{edges } G2$
using $\langle v1' \in \text{nodes } G1 \rangle \langle \text{valid-graph } G2 \rangle$
by (*metis assms(1) disjoint-iff-not-equal valid-graph.E-validD(1)*)
hence $(v1', w, v2) \notin \text{set } ps2$
by (*metis (full-types) assms(4) assms(6) in-mono valid-unMultigraph.path-in-edges*
valid-unMultigraph.is-Eulerian-trail-def)
ultimately show *?thesis using* *distinct-append* **by auto**
qed

moreover have $\text{set } (ps1 @ ((v1', w, v2) \# ps2)) \cap \text{set } (\text{rev-path } (ps1 @ ((v1', w, v2) \# ps2)))$
 $= \{\}$

proof –

have $\text{set } ps1 \cap \text{set } (\text{rev-path } ps1) = \{\}$
by (*metis ps1-G valid valid-unMultigraph.is-trail-path*)
moreover have $\text{set } (\text{rev-path } ps2) \subseteq \text{edges } G2$
by (*metis assms(4) assms(6) valid-unMultigraph.is-trail-rev*
valid-unMultigraph.is-Eulerian-trail-def valid-unMultigraph.path-in-edges)
hence $\text{set } ps1 \cap \text{set } (\text{rev-path } ps2) = \{\}$
using *assms*
valid-unMultigraph.path-in-edges[OF $\langle \text{valid-unMultigraph } G1 \rangle$, *of* $v1$ $ps1$
 $v1$]
valid-unMultigraph.path-in-edges[OF $\langle \text{valid-unMultigraph } G2 \rangle$, *of* $v2$ $ps2$
 $v2$]
unfolding *valid-unMultigraph.is-Eulerian-trail-def*[*OF* $\langle \text{valid-unMultigraph}$
 $G1 \rangle$]
valid-unMultigraph.is-Eulerian-trail-def[*OF* $\langle \text{valid-unMultigraph } G2 \rangle$]
by auto
moreover have $\text{set } ps2 \cap \text{set } (\text{rev-path } ps2) = \{\}$
by (*metis ps2-G valid valid-unMultigraph.is-trail-path*)
moreover have $\text{set } (\text{rev-path } ps1) \subseteq \text{edges } G1$
by (*metis assms(3) assms(5) valid-unMultigraph.is-Eulerian-trail-def*
valid-unMultigraph.path-in-edges valid-unMultigraph.eulerian-rev)
hence $\text{set } ps2 \cap \text{set } (\text{rev-path } ps1) = \{\}$
by (*metis calculation(2) distinct-append distinct-rev-path ps1-G ps2-G*
rev-path-append
rev-path-double valid valid-unMultigraph.is-trail-path)
moreover have $(v2, w, v1') \notin \text{set } (ps1 @ ((v1', w, v2) \# ps2))$

proof –

have $(v2, w, v1') \notin \text{edges } G1$
using $\langle v2 \in \text{nodes } G2 \rangle \langle \text{valid-graph } G1 \rangle$
by (*metis Int-iff all-not-in-conv assms(1) valid-graph.E-validD(1)*)

hence $(v2, w, v1') \notin \text{set } ps1$
by $(metis \text{ assms}(3) \text{ assms}(5) \text{ split-list valid-unMultigraph.is-trail-split'}$
 $\text{ valid-unMultigraph.is-Eulerian-trail-def})$
moreover have $(v2, w, v1') \notin \text{edges } G2$
using $\langle v1' \in \text{nodes } G1 \rangle \langle \text{valid-graph } G2 \rangle$
by $(metis \text{ IntI assms}(1) \text{ empty-iff valid-graph.E-validD}(2))$
hence $(v2, w, v1') \notin \text{set } ps2$
by $(metis (\text{full-types}) \text{ assms}(4) \text{ assms}(6) \text{ in-mono valid-unMultigraph.path-in-edges}$
 $\text{ valid-unMultigraph.is-Eulerian-trail-def})$
moreover have $(v2, w, v1') \neq (v1', w, v2)$
using $\langle v1' \in \text{nodes } G1 \rangle \langle v2 \in \text{nodes } G2 \rangle$
by $(metis \text{ IntI Pair-inject assms}(1) \text{ assms}(5) \text{ be-empty})$
ultimately show $?thesis$ **by** *auto*
qed
ultimately show $?thesis$ **using** *rev-path-append* **by** *auto*
qed
ultimately show $?thesis$ **using** *valid-unMultigraph.is-trail-path*[*OF valid*]
by *auto*
qed
moreover have $\text{edges } (\text{rem-unPath } (ps1 @ ((v1', w, v2) \# ps2)) \text{ } G) = \{\}$
proof –
have $\text{edges } (\text{rem-unPath } (ps1 @ ((v1', w, v2) \# ps2)) \text{ } G) = \text{edges } G -$
 $(\text{set } (ps1 @ ((v1', w, v2) \# ps2)) \cup \text{set } (\text{rev-path } (ps1 @ ((v1', w, v2) \# ps2))))$
by $(metis \text{ rem-unPath-edges})$
also have $\dots = \text{edges } G - (\text{set } ps1 \cup \text{set } ps2 \cup \text{set } (\text{rev-path } ps1) \cup \text{set } (\text{rev-path}$
 $ps2))$
 $\cup \{(v1', w, v2), (v2, w, v1')\}$ **using** *rev-path-append* **by** *auto*
finally have $\text{edges } (\text{rem-unPath } (ps1 @ ((v1', w, v2) \# ps2)) \text{ } G) = \text{edges } G -$
 $(\text{set } ps1 \cup$
 $\text{set } ps2 \cup \text{set } (\text{rev-path } ps1) \cup \text{set } (\text{rev-path } ps2) \cup \{(v1', w, v2), (v2, w, v1')\})$
 $.$
moreover have $\text{edges } (\text{rem-unPath } ps1 \text{ } G1) = \{\}$
by $(metis \text{ assms}(3) \text{ assms}(5) \text{ valid-unMultigraph.is-Eulerian-trail-def})$
hence $\text{edges } G1 - (\text{set } ps1 \cup \text{set } (\text{rev-path } ps1)) = \{\}$
by $(metis \text{ rem-unPath-edges})$
moreover have $\text{edges } (\text{rem-unPath } ps2 \text{ } G2) = \{\}$
by $(metis \text{ assms}(4) \text{ assms}(6) \text{ valid-unMultigraph.is-Eulerian-trail-def})$
hence $\text{edges } G2 - (\text{set } ps2 \cup \text{set } (\text{rev-path } ps2)) = \{\}$
by $(metis \text{ rem-unPath-edges})$
ultimately show $?thesis$ **using** *G* **by** *auto*
qed
ultimately show $?thesis$ **by** $(metis \text{ } G \text{ valid valid-unMultigraph.is-Eulerian-trail-def})$
qed
lemma (in *valid-unMultigraph*) *eulerian-sufficient*:
assumes *finite V finite E connected V ≠ { }*
shows *num-of-odd-nodes G = 2 ⇒*
 $(\exists v \in V. \exists v' \in V. \exists ps. \text{odd}(\text{degree } v \text{ } G) \wedge \text{odd}(\text{degree } v' \text{ } G) \wedge (v \neq v') \wedge \text{is-Eulerian-trail}$
 $v \text{ } ps \text{ } v')$

and $\text{num-of-odd-nodes } G=0 \implies (\forall v \in V. \exists ps. \text{is-Eulerian-circuit } v \text{ ps } v)$
using $\langle \text{finite } E \rangle \langle \text{finite } V \rangle \text{valid-unMultigraph-axioms } \langle V \neq \{\} \rangle \langle \text{connected} \rangle$
proof (*induct card E arbitrary: G rule: less-induct*)
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** $\text{num-of-odd-nodes } G = 2$
have $\text{valid-graph } G$ **using** $\langle \text{valid-unMultigraph } G \rangle \text{valid-unMultigraph-def}$ **by**
auto
obtain $n1 \ n2$ **where**
 $n1: n1 \in \text{nodes } G \text{ odd}(\text{degree } n1 \ G)$
and $n2: n2 \in \text{nodes } G \text{ odd}(\text{degree } n2 \ G)$
and $n1 \neq n2$ **unfolding** $\text{num-of-odd-nodes-def odd-nodes-set-def}$
proof –
have $\forall S. \text{card } S=2 \implies (\exists n1 \ n2. n1 \in S \wedge n2 \in S \wedge n1 \neq n2)$
by (*metis card-eq-0-iff equals0I even-card' even-numeral zero-neq-numeral*)
then obtain $t1 \ t2$
where $t1 \in \{v \in \text{nodes } G. \text{odd}(\text{degree } v \ G)\}$ $t2 \in \{v \in \text{nodes } G. \text{odd}(\text{degree } v \ G)\}$ $t1 \neq t2$
using $\langle \text{num-of-odd-nodes } G = 2 \rangle$ **unfolding** $\text{num-of-odd-nodes-def odd-nodes-set-def}$
by *force*
thus *?thesis* **by** (*metis (lifting) that mem-Collect-eq*)
qed
have $\text{even-except-two: } \bigwedge n. n \in \text{nodes } G \implies n \neq n1 \implies n \neq n2 \implies \text{even}(\text{degree } n \ G)$
proof (*rule ccontr*)
fix n **assume** $n \in \text{nodes } G \ n \neq n1 \ n \neq n2 \text{ odd}(\text{degree } n \ G)$
have $n \in \text{odd-nodes-set } G$
by (*metis (mono-tags) $\langle n \in \text{nodes } G \rangle \langle \text{odd}(\text{degree } n \ G) \rangle \text{mem-Collect-eq odd-nodes-set-def}$*)
moreover **have** $n1 \in \text{odd-nodes-set } G$
by (*metis (mono-tags) mem-Collect-eq n1(1) n1(2) odd-nodes-set-def*)
moreover **have** $n2 \in \text{odd-nodes-set } G$
using $n2(1) \ n2(2)$ **unfolding** odd-nodes-set-def **by** *auto*
ultimately **have** $\{n, n1, n2\} \subseteq \text{odd-nodes-set } G$ **by** *auto*
moreover **have** $\text{card}\{n, n1, n2\} \geq 3$ **using** $\langle n1 \neq n2 \rangle \langle n \neq n1 \rangle \langle n \neq n2 \rangle$ **by** *auto*
moreover **have** $\text{finite } (\text{odd-nodes-set } G)$
using $\langle \text{finite } (\text{nodes } G) \rangle$ **unfolding** odd-nodes-set-def **by** *auto*
ultimately **have** $\text{card } (\text{odd-nodes-set } G) \geq 3$
using $\text{card-mono}[of \ \text{odd-nodes-set } G \ \{n, n1, n2\}]$ **by** *auto*
thus *False* **using** $\langle \text{num-of-odd-nodes } G = 2 \rangle$ **unfolding** $\text{num-of-odd-nodes-def}$
by *auto*
qed
have $\{e \in \text{edges } G. \text{fst } e = n1\} \neq \{\}$
using $n1$
by (*metis (full-types) degree-def empty-iff finite.emptyI odd-card*)
then obtain $v' \ w$ **where** $(n1, w, v') \in \text{edges } G$ **by** *auto*
have $v' = n2 \implies (\exists v \in \text{nodes } G. \exists v' \in \text{nodes } G. \exists ps. \text{odd}(\text{degree } v \ G) \wedge \text{odd}(\text{degree } v' \ G) \wedge v \neq v')$

```

     $\wedge$  valid-unMultigraph.is-Eulerian-trail  $G$   $v$   $ps$   $v'$ )
  proof (cases valid-unMultigraph.connected (del-unEdge  $n1$   $w$   $n2$   $G$ ))
    assume  $v'=n2$ 
    assume conneted':valid-unMultigraph.connected (del-unEdge  $n1$   $w$   $n2$   $G$ )
    moreover have num-of-odd-nodes (del-unEdge  $n1$   $w$   $n2$   $G$ ) = 0
      using  $\langle n1, w, v' \rangle \in$  edges  $G$   $\langle$  finite (edges  $G$ )  $\rangle$   $\langle$  finite (nodes  $G$ )  $\rangle$   $\langle v' =$ 
 $n2 \rangle$ 
       $\langle$  num-of-odd-nodes  $G = 2 \rangle$   $\langle$  valid-unMultigraph  $G \rangle$  del-UnEdge-odd-odd
 $n1(2)$   $n2(2)$ 
      by force
    moreover have finite (edges (del-unEdge  $n1$   $w$   $n2$   $G$ ))
      using  $\langle$  finite (edges  $G$ )  $\rangle$  by auto
    moreover have finite (nodes (del-unEdge  $n1$   $w$   $n2$   $G$ ))
      using  $\langle$  finite (nodes  $G$ )  $\rangle$  by auto
    moreover have edges  $G - \{(n1,w,n2),(n2,w,n1)\} \subset$  edges  $G$ 
      using Diff-iff Diff-subset  $\langle (n1, w, v') \in$  edges  $G \rangle$   $\langle v' = n2 \rangle$ 
      by fast
    hence card (edges (del-unEdge  $n1$   $w$   $n2$   $G$ )) < card (edges  $G$ )
    using  $\langle$  finite (edges  $G$ )  $\rangle$  psubset-card-mono[of edges  $G$  edges  $G - \{(n1,w,n2),(n2,w,n1)\}$ ]
      unfolding del-unEdge-def by auto
    moreover have valid-unMultigraph (del-unEdge  $n1$   $w$   $n2$   $G$ )
      using  $\langle$  valid-unMultigraph  $G \rangle$  del-unEdge-valid by auto
    moreover have nodes (del-unEdge  $n1$   $w$   $n2$   $G$ )  $\neq$   $\{\}$ 
      by (metis (full-types) del-UnEdge-node empty-iff  $n1(1)$ )
    ultimately have  $\forall v \in$  nodes (del-unEdge  $n1$   $w$   $n2$   $G$ ).  $\exists ps.$  valid-unMultigraph.is-Eulerian-circuit
      (del-unEdge  $n1$   $w$   $n2$   $G$ )  $v$   $ps$   $v$ 
      using less.hyps[of del-unEdge  $n1$   $w$   $n2$   $G$ ] by auto
    thus ?thesis using eulerian-cons
      by (metis  $\langle (n1, w, v') \in$  edges  $G \rangle$   $\langle n1 \neq n2 \rangle$   $\langle v' = n2 \rangle$   $\langle$  valid-unMultigraph
 $G \rangle$ 
       $\langle$  valid-unMultigraph (del-unEdge  $n1$   $w$   $n2$   $G$ )  $\rangle$  del-UnEdge-node  $n1(1)$ 
 $n1(2)$   $n2(1)$   $n2(2)$ 
      valid-unMultigraph.eulerian-cons valid-unMultigraph.is-Eulerian-circuit-def)
    next
    assume  $v'=n2$ 
    assume not-conneted: $\neg$ valid-unMultigraph.connected (del-unEdge  $n1$   $w$   $n2$   $G$ )
    have valid0:valid-unMultigraph (del-unEdge  $n1$   $w$   $n2$   $G$ )
      using  $\langle$  valid-unMultigraph  $G \rangle$  del-unEdge-valid by auto
    hence valid0':valid-graph (del-unEdge  $n1$   $w$   $n2$   $G$ )
      using valid-unMultigraph-def by auto
    have all-even: $\forall n \in$  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  $\langle (n1, w, v') \in$  edges  $G \rangle$   $\langle$  finite (edges  $G$ )  $\rangle$   $\langle v' = n2 \rangle$   $\langle$  valid-unMultigraph
 $G \rangle$   $n1$ 
      by (auto simp add: valid-unMultigraph.corres)
    moreover have even (degree  $n2$  (del-unEdge  $n1$   $w$   $n2$   $G$ ))
      using  $\langle (n1, w, v') \in$  edges  $G \rangle$   $\langle$  finite (edges  $G$ )  $\rangle$   $\langle v' = n2 \rangle$   $\langle$  valid-unMultigraph

```

$G \triangleright n2$
by (*auto simp add: valid-unMultigraph.corres*)
moreover have $\bigwedge n. n \in \text{nodes } (\text{del-unEdge } n1 \ w \ n2 \ G) \implies n \neq n1 \implies$
 $n \neq n2 \implies$
even (*degree* n (*del-unEdge* $n1 \ w \ n2 \ G$))
using *valid-unMultigraph.degree-frame*[*OF* $\langle \text{valid-unMultigraph } G \rangle$,
of - n1 n2 w] *even-except-two*
by (*metis* (*no-types*) $\langle \text{finite } (\text{edges } G) \rangle$ *del-unEdge-def empty-iff insert-iff*
select-convs(1))
ultimately show *?thesis* **by auto**
qed
have $(n1, w, n2) \in \text{edges } G$ **by** (*metis* $\langle (n1, w, v') \in \text{edges } G \rangle \langle v' = n2 \rangle$)
hence $(n2, w, n1) \in \text{edges } G$ **by** (*metis* $\langle \text{valid-unMultigraph } G \rangle$ *valid-unMultigraph.corres*)
obtain $G1 \ G2$ **where**
 $G1\text{-nodes: nodes } G1 = \{n. \exists ps. \text{valid-graph.is-path } (\text{del-unEdge } n1 \ w \ n2 \ G)$
 $n \ ps \ n1\}$
and $G1\text{-edges: edges } G1 = \{(n, e, n'). (n, e, n') \in \text{edges } (\text{del-unEdge } n1 \ w \ n2$
 $G)\}$
 $\wedge n \in \text{nodes } G1 \wedge n' \in \text{nodes } G1\}$
and $G2\text{-nodes: nodes } G2 = \{n. \exists ps. \text{valid-graph.is-path } (\text{del-unEdge } n1 \ w$
 $n2 \ G) \ n \ ps \ n2\}$
and $G2\text{-edges: edges } G2 = \{(n, e, n'). (n, e, n') \in \text{edges } (\text{del-unEdge } n1 \ w \ n2 \ G)$
 $\wedge n \in \text{nodes } G2$
 $\wedge n' \in \text{nodes } G2\}$
and $G1\text{-}G2\text{-edges-union: edges } G1 \cup \text{edges } G2 = \text{edges } (\text{del-unEdge } n1 \ w$
 $n2 \ G)$
and $\text{edges } G1 \cap \text{edges } G2 = \{\}$
and $G1\text{-}G2\text{-nodes-union: nodes } G1 \cup \text{nodes } G2 = \text{nodes } (\text{del-unEdge } n1 \ w$
 $n2 \ G)$
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* $G2$
using *valid-unMultigraph.connectivity-split*[*OF* $\langle \text{valid-unMultigraph } G \rangle$
 $\langle \text{valid-unMultigraph.connected } G \rangle \langle \neg \text{valid-unMultigraph.connected } (\text{del-unEdge } n1 \ w \ n2 \ G) \rangle$
 $\langle (n1, w, n2) \in \text{edges } G \rangle]$.
have $\text{edges } (\text{del-unEdge } n1 \ w \ n2 \ G) \subset \text{edges } G$
unfolding *del-unEdge-def* **using** $\langle (n1, w, n2) \in \text{edges } G \rangle \langle (n2, w, n1) \in \text{edges } G \rangle$
by auto
hence $\text{card } (\text{edges } G1) < \text{card } (\text{edges } G)$ **using** $G1\text{-}G2\text{-edges-union}$
by (*metis* (*full-types*) $\langle \text{finite } (\text{edges } G) \rangle$ *inf-sup-absorb less-infl2* *psub-*
set-card-mono)
moreover have *finite* (*edges* $G1$)
using $G1\text{-}G2\text{-edges-union } \langle \text{finite } (\text{edges } G) \rangle$
by (*metis* $\langle \text{edges } (\text{del-unEdge } n1 \ w \ n2 \ G) \subset \text{edges } G \rangle$ *finite-Un less-imp-le*
rev-finite-subset)
moreover have $\text{nodes } G1 \subseteq \text{nodes } (\text{del-unEdge } n1 \ w \ n2 \ G)$

```

    by (metis G1-G2-nodes-union Un-upper1)
  hence finite (nodes G1)
    using ⟨finite (nodes G)⟩ del-UnEdge-node rev-finite-subset by auto
  moreover have n1 ∈ nodes G1
  proof -
    have n1 ∈ nodes (del-unEdge n1 w n2 G) using ⟨n1 ∈ nodes G⟩ by auto
    hence valid-graph.is-path (del-unEdge n1 w n2 G) n1 [] n1
      using valid0' by (metis valid-graph.is-path-simps(1))
    thus ?thesis using G1-nodes by auto
  qed
  hence nodes G1 ≠ {} by auto
  moreover have num-of-odd-nodes G1 = 0
  proof -
    have valid-graph G2 using ⟨valid-unMultigraph G2⟩ valid-unMultigraph-def
  by auto
    hence ∀ n ∈ nodes G1. degree n G1 = degree n (del-unEdge n1 w n2 G)
    using sub-graph-degree-frame[of G2 G1 (del-unEdge n1 w n2 G)]
      by (metis G1-G2-edges-union ⟨nodes G1 ∩ nodes G2 = {}⟩)
    hence ∀ n ∈ nodes G1. even(degree n G1) 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 G1. ∃ ps. valid-unMultigraph.is-Eulerian-circuit
    G1 v ps v
    using less.hyps[of G1] ⟨valid-unMultigraph G1⟩ ⟨valid-unMultigraph.connected
    G1⟩
      by auto
  then obtain ps1 where ps1: valid-unMultigraph.is-Eulerian-trail G1 n1 ps1
  n1
    using ⟨n1 ∈ nodes G1⟩
  by (metis (full-types) ⟨valid-unMultigraph G1⟩ valid-unMultigraph.is-Eulerian-circuit-def)
  have card (edges G2) < card (edges G)
    using G1-G2-edges-union ⟨edges (del-unEdge n1 w n2 G) ⊂ edges G⟩
      by (metis (full-types) ⟨finite (edges G)⟩ inf-sup-ord(4) le-less-trans psub-
    set-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
        unfolding del-unEdge-def by auto
      ultimately have (|nodes = nodes (del-unEdge n1 w n2 G), edges =
        edges (del-unEdge n1 w n2 G) ∪ {(n1, w, n2), (n2, w, n1)}})= G
        by auto
      moreover have (|nodes = nodes G1 ∪ nodes G2, edges = edges G1 ∪
edges G2 ∪
  {(n1, w, n2),(n2, w, n1)}})=(|nodes = nodes (del-unEdge n1 w n2
G),edges
  = edges (del-unEdge n1 w n2 G) ∪ {(n1, w, n2), (n2, w, n1)}})
        by (metis G1-G2-edges-union G1-G2-nodes-union)
      ultimately show ?thesis by auto
    qed
  qed

```

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

```


moreover have $v' \neq n1$
using *valid-unMultigraph.no-id*[*OF* $\langle \text{valid-unMultigraph } G \rangle$] $\langle (n1, w, v') \in \text{edges } G \rangle$ **by auto**
hence $\text{odd-}v': \text{odd}(\text{degree } v' (\text{del-unEdge } n1 \ w \ v' \ G))$
using $\langle v' \neq n2 \rangle$ *even-except-two*[*of* v']
valid-graph.E-validD(2)[*OF* $\langle \text{valid-graph } G \rangle$] $\langle (n1, w, v') \in \text{edges } G \rangle$
valid-unMultigraph.del-UnEdge-even'[*OF* $\langle \text{valid-unMultigraph } G \rangle$] $\langle (n1, w, v') \in \text{edges } G \rangle$
 $\langle \text{finite } (\text{edges } G) \rangle$]
unfolding *odd-nodes-set-def* **by auto**
ultimately have $\text{even-except-two}': \bigwedge n. n \in \text{nodes } (\text{del-unEdge } n1 \ w \ v' \ G) \implies$
 $n \neq n2 \implies n \neq v' \implies \text{even}(\text{degree } n (\text{del-unEdge } n1 \ w \ v' \ G))$
using *del-UnEdge-node*[*of* $n1 \ w \ v' \ G$] *even-except-two* *valid-unMultigraph.degree-frame*[*OF*
 $\langle \text{valid-unMultigraph } G \rangle$ $\langle \text{finite } (\text{edges } G) \rangle$, *of* $n1 \ v' \ w$]
by force
obtain $G1 \ G2$ **where**
 $G1\text{-nodes: nodes } G1 = \{n. \exists ps. \text{valid-graph.is-path } (\text{del-unEdge } n1 \ w \ v' \ G)$
 $n \ ps \ n1\}$
and $G1\text{-edges: edges } G1 = \{(n, e, n'). (n, e, n') \in \text{edges } (\text{del-unEdge } n1 \ w \ v' \ G)$
 $\wedge n \in \text{nodes } G1$
 $\wedge n' \in \text{nodes } G1\}$
and $G2\text{-nodes: nodes } G2 = \{n. \exists ps. \text{valid-graph.is-path } (\text{del-unEdge } n1 \ w \ v'$
 $G) \ n \ ps \ v'\}$
and $G2\text{-edges: edges } G2 = \{(n, e, n'). (n, e, n') \in \text{edges } (\text{del-unEdge } n1 \ w \ v' \ G)$
 $\wedge n \in \text{nodes } G2$
 $\wedge n' \in \text{nodes } G2\}$
and $G1\text{-}G2\text{-edges-union: edges } G1 \cup \text{edges } G2 = \text{edges } (\text{del-unEdge } n1 \ w$
 $v' \ G)$
and $\text{edges } G1 \cap \text{edges } G2 = \{\}$
and $G1\text{-}G2\text{-nodes-union: nodes } G1 \cup \text{nodes } G2 = \text{nodes } (\text{del-unEdge } n1 \ w$
 $v' \ G)$
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* $G2$
using *valid-unMultigraph.connectivity-split*[*OF* $\langle \text{valid-unMultigraph } G \rangle$
 $\langle \text{valid-unMultigraph.connected } G \rangle$ *not-conneted* $\langle (n1, w, v') \in \text{edges } G \rangle$]
 \cdot
have $n2 \in \text{nodes } G2$ **using** *extend-distinct-path*
proof –
have *finite* ($\text{edges } (\text{del-unEdge } n1 \ w \ v' \ G)$)
unfolding *del-unEdge-def* **using** $\langle \text{finite } (\text{edges } G) \rangle$ **by auto**
moreover have $\text{num-of-odd-nodes } (\text{del-unEdge } n1 \ w \ v' \ G) = 2$
by (*metis* $\langle (n1, w, v') \in \text{edges } G \rangle$ $\langle (v', w, n1) \in \text{edges } G \rangle$ $\langle \text{num-of-odd-nodes}$
 $G = 2 \rangle$
 $\langle v' \neq n2 \rangle$ $\langle \text{valid-graph } G \rangle$ *del-UnEdge-even-odd delete-edge-sym*
even-except-two

$\langle \text{finite (edges } G \rangle \langle \text{finite (nodes } G \rangle \langle \text{valid-unMultigraph } G \rangle$
 $n1(2) \text{ valid-graph.E-validD}(2) \text{ valid-unMultigraph.no-id}$
ultimately have $\exists ps. \text{ valid-unMultigraph.is-trail (del-unEdge } n1 \text{ } w \text{ } v' \text{ } G)$
 $n2 \text{ } ps \text{ } v'$
using $\text{ valid-unMultigraph.path-between-odds}[OF \text{ valid0, of } n2 \text{ } v', OF \text{ odd-n2}$
 $\text{ odd-v'}] \langle v' \neq n2 \rangle$
by auto
hence $\exists ps. \text{ valid-graph.is-path (del-unEdge } n1 \text{ } w \text{ } v' \text{ } G) \text{ } n2 \text{ } ps \text{ } v'$
by $(\text{metis valid0 valid-unMultigraph.is-trail-intro})$
thus ?thesis using G2-nodes by auto
qed
have $v' \in \text{nodes } G2$
proof –
have $\text{ valid-graph.is-path (del-unEdge } n1 \text{ } w \text{ } v' \text{ } G) \text{ } v' \text{ } [] \text{ } v'$
by $(\text{metis (full-types) } \langle n1, w, v' \rangle \in \text{edges } G \rangle \langle \text{valid-graph } G \rangle \text{ del-UnEdge-node}$
 $\text{ valid0' valid-graph.E-validD}(2) \text{ valid-graph.is-path-simps}(1))$
thus ?thesis by (metis (lifting) G2-nodes mem-Collect-eq)
qed
have $\text{edges-subset:edges (del-unEdge } n1 \text{ } w \text{ } v' \text{ } G) \subset \text{edges } G$
using $\langle (n1, w, v') \in \text{edges } G \rangle \langle (v', w, n1) \in \text{edges } G \rangle$
unfolding del-unEdge-def by auto
hence $\text{card (edges } G1) < \text{card (edges } G)$
by $(\text{metis } G1\text{-}G2\text{-edges-union inf-sup-absorb } \langle \text{finite (edges } G \rangle \text{ } \text{less-infI2}$
 $\text{psubset-card-mono})$
moreover have $\text{finite (edges } G1)$
by $(\text{metis (full-types) } G1\text{-}G2\text{-edges-union edges-subset finite-Un finite-subset}$
 $\langle \text{finite (edges } G \rangle \text{ } \text{less-imp-le})$
moreover have $\text{finite (nodes } G1)$
using $G1\text{-}G2\text{-nodes-union } \langle \text{finite (nodes } G \rangle$
unfolding del-unEdge-def
by $(\text{metis (full-types) finite-Un select-convs}(1))$
moreover have $n1 \in \text{nodes } G1$
proof –
have $\text{ valid-graph.is-path (del-unEdge } n1 \text{ } w \text{ } v' \text{ } G) \text{ } n1 \text{ } [] \text{ } n1$
by $(\text{metis (full-types) del-UnEdge-node } n1(1) \text{ valid0' valid-graph.is-path-simps}(1))$
thus ?thesis by (metis (lifting) G1-nodes mem-Collect-eq)
qed
moreover hence $\text{nodes } G1 \neq \{\}$ **by auto**
moreover have $\text{num-of-odd-nodes } G1 = 0$
proof –
have $\forall n \in \text{nodes } G1. \text{even}(\text{degree } n \text{ (del-unEdge } n1 \text{ } w \text{ } v' \text{ } G))$
using $\text{even-except-two' odd-v' odd-n2 } \langle n2 \in \text{nodes } G2 \rangle \langle \text{nodes } G1 \cap \text{nodes}$
 $G2 = \{\}$
 $\langle v' \in \text{nodes } G2 \rangle$
by $(\text{metis (full-types) } G1\text{-}G2\text{-nodes-union Un-iff disjoint-iff-not-equal})$
moreover have $\text{valid-graph } G2$
using $\langle \text{valid-unMultigraph } G2 \rangle \text{ valid-unMultigraph-def}$
by auto
ultimately have $\forall n \in \text{nodes } G1. \text{even}(\text{degree } n \text{ } G1)$

```

    using sub-graph-degree-frame[of G2 G1 del-unEdge n1 w v' G]
    by (metis G1-G2-edges-union ⟨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-nodes-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 = {}⟩)
    hence ∀ n∈nodes G2. n∉{n2,v'}⟶n∉{v ∈ nodes G2. odd (degree v G2)}
      by (metis (lifting) mem-Collect-eq)
    moreover have odd(degree n2 G2)
      using sub-graph-degree-frame[of G1 G2 del-unEdge n1 w v' G]
      by (metis (opaque-lifting, no-types) G1-G2-edges-union ⟨nodes G1 ∩
nodes G2 = {}⟩
        ⟨valid-graph G1⟩ ⟨n2 ∈ nodes G2⟩ inf-assoc inf-bot-right inf-sup-absorb
        odd-n2 sup-bot-right sup-commute)
    hence n2∈{v ∈ nodes G2. odd (degree v G2)}
      by (metis (lifting) ⟨n2 ∈ nodes G2⟩ mem-Collect-eq)
    moreover have odd(degree v' G2)
      using sub-graph-degree-frame[of G1 G2 del-unEdge n1 w v' G]
      by (metis G1-G2-edges-union Int-commute Un-commute ⟨nodes G1 ∩
nodes G2 = {}⟩
        ⟨v' ∈ nodes G2⟩ ⟨valid-graph G1⟩ odd-v')

```

hence $v' \in \{v \in \text{nodes } G2. \text{ odd } (\text{degree } v \ G2)\}$
by (*metis* (*full-types*) *Collect-conj-eq* *Collect-mem-eq* *Int-Collect* $\langle v' \in \text{nodes } G2 \rangle$)
ultimately have $\{v \in \text{nodes } G2. \text{ odd } (\text{degree } v \ G2)\} = \{n2, v'\}$
using $\langle \text{finite } (\text{nodes } G2) \rangle$ **by** (*induct* *G2, auto*)
thus *?thesis* **using** $\langle v' \neq n2 \rangle$
unfolding *num-of-odd-nodes-def* *odd-nodes-set-def* **by** *auto*
qed
ultimately obtain $s \ t \ \text{ps2}$ **where**
 $s: s \in \text{nodes } G2 \ \text{odd } (\text{degree } s \ G2)$
and $t: t \in \text{nodes } G2 \ \text{odd } (\text{degree } t \ G2)$
and $s \neq t$
and $s\text{-ps2-t}: \text{valid-unMultigraph.is-Eulerian-trail } G2 \ s \ \text{ps2} \ t$
using $\langle \text{valid-unMultigraph } G2 \rangle \langle \text{valid-unMultigraph.connected } G2 \rangle$ *less.hyps*[*of*
G2]
by *auto*
moreover have *valid-graph* *G1*
using $\langle \text{valid-unMultigraph } G1 \rangle$ *valid-unMultigraph-def* **by** *auto*
ultimately have $(s = n2 \wedge t = v') \vee (s = v' \wedge t = n2)$
using *odd-n2* *odd-v'* *even-except-two'*
sub-graph-degree-frame[*of* *G1* *G2* (*del-unEdge* $n1 \ w \ v' \ G$)]
by (*metis* *G1-G2-edges-union* *G1-G2-nodes-union* *UnI1* $\langle \text{nodes } G1 \cap \text{nodes } G2 = \{\} \rangle$ *inf-commute*
sup commute)
moreover have *merge-G1-G2*: $(\text{nodes} = \text{nodes } G1 \cup \text{nodes } G2, \text{edges} = \text{edges } G1 \cup \text{edges } G2 \cup$
 $\{(n1, w, v'), (v', w, n1)\}) = G$
proof –
have $\text{edges } (\text{del-unEdge } n1 \ w \ v' \ G) \cup \{(n1, w, v'), (v', w, n1)\} = \text{edges } G$
using $\langle (n1, w, v') \in \text{edges } G \rangle \langle (v', w, n1) \in \text{edges } G \rangle$
unfolding *del-unEdge-def* **by** *auto*
moreover have $\text{nodes } (\text{del-unEdge } n1 \ w \ v' \ G) = \text{nodes } G$
unfolding *del-unEdge-def* **by** *auto*
ultimately have $(\text{nodes} = \text{nodes } (\text{del-unEdge } n1 \ w \ v' \ G), \text{edges} =$
 $\text{edges } (\text{del-unEdge } n1 \ w \ v' \ G) \cup \{(n1, w, v'), (v', w, n1)\}) = G$
by *auto*
moreover have $(\text{nodes} = \text{nodes } G1 \cup \text{nodes } G2, \text{edges} = \text{edges } G1 \cup$
 $\text{edges } G2 \cup$
 $\{(n1, w, v'), (v', w, n1)\}) = (\text{nodes} = \text{nodes } (\text{del-unEdge } n1 \ w \ v' \ G), \text{edges}$
 $= \text{edges } (\text{del-unEdge } n1 \ w \ v' \ G) \cup \{(n1, w, v'), (v', w, n1)\})$
by (*metis* *G1-G2-edges-union* *G1-G2-nodes-union*)
ultimately show *?thesis* **by** *auto*
qed
moreover have $s = n2 \implies t = v' \implies ?thesis$
using *eulerian-split*[*of* *G1* *G2* $n1 \ \text{ps1} \ n1 \ v' \ (\text{rev-path } \text{ps2}) \ n2 \ w$] *merge-G1-G2*
by (*metis* $\langle \text{edges } G1 \cap \text{edges } G2 = \{\} \rangle \langle n1 \neq n2 \rangle \langle \text{nodes } G1 \cap \text{nodes } G2 = \{\} \rangle$
 $\langle \text{valid-unMultigraph } G1 \rangle \langle \text{valid-unMultigraph } G2 \rangle \ n1(1) \ n1(2) \ n2(1) \ n2(2) \ \text{ps1} \ s\text{-ps2-t}$)

```

      valid-unMultigraph.euclerian-rev)
moreover have  $s=v' \implies t=n2 \implies ?thesis$ 
  using eulerian-split[of G1 G2 n1 ps1 n1 v' ps2 n2 w] merge-G1-G2
  by (metis ⟨edges G1 ∩ edges G2 = {}⟩ ⟨n1 ≠ n2⟩ ⟨nodes G1 ∩ nodes G2
= {}⟩
      ⟨valid-unMultigraph G1⟩ ⟨valid-unMultigraph G2⟩ n1(1) n1(2) n2(1)
n2(2) ps1 s-ps2-t)
  ultimately show ?thesis by auto
qed
ultimately show  $\exists v \in \text{nodes } G. \exists v' \in \text{nodes } G. \exists ps. \text{odd}(\text{degree } v \ G) \wedge \text{odd}(\text{degree } v' \ G) \wedge v \neq v'$ 
   $\wedge \text{valid-unMultigraph.is-Eulerian-trail } G \ v \ ps \ v'$ 
by auto
next
  case less
    assume finite (edges G) and finite (nodes G) and valid-unMultigraph G and
nodes G ≠ {}
    and valid-unMultigraph.connected G and num-of-odd-nodes G = 0
  show  $\forall v \in \text{nodes } G. \exists ps. \text{valid-unMultigraph.is-Eulerian-circuit } G \ v \ ps \ v$ 
  proof (rule,cases card (nodes G)=1)
    fix v assume  $v \in \text{nodes } G$ 
    assume card (nodes G) = 1
    hence nodes G = {v}
    using ⟨v ∈ nodes G⟩ card-Suc-eq[of nodes G 0] empty-iff insert-iff[of - v]
    by auto
    have edges G = {}
    proof (rule ccontr)
      assume edges G ≠ {}
      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 ⟨nodes G={v}⟩
      by (metis (opaque-lifting, no-types) append-Nil2 valid-unMultigraph.is-trail-rev
valid-unMultigraph.is-trail.simps(1) ⟨valid-unMultigraph G⟩ singletonE
valid-unMultigraph.is-trail-split valid-unMultigraph.singleton-distinct-path)
      thus False by (metis e ⟨valid-unMultigraph G⟩ valid-unMultigraph.no-id)
    qed
    hence valid-unMultigraph.is-Eulerian-circuit G v [] v
  by (metis ⟨nodes G = {v}⟩ insert-subset ⟨valid-unMultigraph G⟩ rem-unPath.simps(1)
subsetI valid-unMultigraph.is-trail.simps(1)
valid-unMultigraph.is-Eulerian-circuit-def
valid-unMultigraph.is-Eulerian-trail-def)
  thus  $\exists ps. \text{valid-unMultigraph.is-Eulerian-circuit } G \ v \ ps \ v$  by auto
next
  fix v assume  $v \in \text{nodes } G$ 
  assume card (nodes G) ≠ 1
  moreover have card (nodes G) ≠ 0 using ⟨nodes G ≠ {}⟩
    by (metis card-eq-0-iff ⟨finite (nodes G)⟩)
  ultimately have card (nodes G) ≥ 2 by auto
  then obtain n where card (nodes G) = Suc (Suc n)

```

by (*metis le-iff-add add-2-eq-Suc*)
hence $\exists n \in \text{nodes } G. n \neq v$ **by** (*auto dest!: card-eq-SucD*)
then obtain $v' w$ **where** $(v, w, v') \in \text{edges } G$
proof –
assume $\text{pre}: \bigwedge w v'. (v, w, v') \in \text{edges } G \implies \text{thesis}$
assume $\exists n \in \text{nodes } G. n \neq v$
then obtain ps **where** $ps: \exists v'. \text{valid-graph.is-path } G v ps v' \wedge ps \neq \text{Nil}$
using *valid-unMultigraph-def*
by (*metis (full-types) $\langle v \in \text{nodes } G \rangle \langle \text{valid-unMultigraph } G \rangle \text{valid-graph.is-path.simps}(1)$*
 $\langle \text{valid-unMultigraph.connected } G \rangle \text{valid-unMultigraph.connected-def}$)
then obtain $v0 w v'$ **where** $\exists ps'. ps = \text{Cons } (v0, w, v') ps'$ **by** (*metis*
neq-Nil-conv prod-cases3)
hence $v0 = v$
using *valid-unMultigraph-def*
by (*metis $\langle \text{valid-unMultigraph } G \rangle ps \text{valid-graph.is-path.simps}(2)$*)
hence $(v, w, v') \in \text{edges } G$
using *valid-unMultigraph-def*
by (*metis $\langle \exists ps'. ps = (v0, w, v') \# ps' \rangle \langle \text{valid-unMultigraph } G \rangle ps$*
 $\text{valid-graph.is-path.simps}(2)$)
thus *?thesis* **by** (*metis pre*)
qed
have *all-even*: $\forall x \in \text{nodes } G. \text{even}(\text{degree } x G)$
using $\langle \text{finite } (\text{nodes } G) \rangle \langle \text{num-of-odd-nodes } G = 0 \rangle$
unfolding *num-of-odd-nodes-def odd-nodes-set-def* **by** *auto*
have *odd-v*: $\text{odd}(\text{degree } v (\text{del-unEdge } v w v' G))$
using $\langle v \in \text{nodes } G \rangle \text{all-even } \text{valid-unMultigraph.del-UnEdge-even}[OF$
 $\langle \text{valid-unMultigraph } G \rangle$
 $\langle (v, w, v') \in \text{edges } G \rangle \langle \text{finite } (\text{edges } G) \rangle]$
unfolding *odd-nodes-set-def* **by** *auto*
have *odd-v'*: $\text{odd}(\text{degree } v' (\text{del-unEdge } v w v' G))$
using *valid-unMultigraph.del-UnEdge-even'[OF $\langle \text{valid-unMultigraph } G \rangle \langle (v,$*
 $w, v') \in \text{edges } G \rangle$
 $\langle \text{finite } (\text{edges } G) \rangle]$
 $\text{all-even } \text{valid-graph.E-validD}(2)[OF - \langle (v, w, v') \in \text{edges } G \rangle$
 $\langle \text{valid-unMultigraph } G \rangle$
unfolding *valid-unMultigraph-def odd-nodes-set-def*
by *auto*
have *valid-unMulti*: *valid-unMultigraph* (*del-unEdge* $v w v' G$)
by (*metis del-unEdge-valid $\langle \text{valid-unMultigraph } G \rangle$*)
moreover **have** *valid-graph*: *valid-graph* (*del-unEdge* $v w v' G$)
using *valid-unMultigraph-def del-undirected*
by (*metis $\langle \text{valid-unMultigraph } G \rangle \text{delete-edge-valid}$*)
moreover **have** *fin-E'*: $\text{finite}(\text{edges } (\text{del-unEdge } v w v' G))$
using $\langle \text{finite}(\text{edges } G) \rangle$ **unfolding** *del-unEdge-def* **by** *auto*
moreover **have** *fin-V'*: $\text{finite}(\text{nodes } (\text{del-unEdge } v w v' G))$
using $\langle \text{finite}(\text{nodes } G) \rangle$ **unfolding** *del-unEdge-def* **by** *auto*
moreover **have** *less-card*: $\text{card}(\text{edges } (\text{del-unEdge } v w v' G)) < \text{card}(\text{edges } G)$
unfolding *del-unEdge-def* **using** $\langle (v, w, v') \in \text{edges } G \rangle$
by (*metis Diff-insert2 card-Diff2-less $\langle \text{finite } (\text{edges } G) \rangle \langle \text{valid-unMultigraph}$*

```

G>
  select-conv(2) valid-unMultigraph.corres)
moreover have num-of-odd-nodes (del-unEdge v w v' G) = 2
  using <valid-unMultigraph G> <num-of-odd-nodes G = 0> <v ∈ nodes G>
all-even
  del-UnEdge-even-even[OF <valid-unMultigraph G> <finite (edges G)> <finite
(nodes G)>
  <(v, w, v') ∈ edges G>] valid-graph.E-validD(2)[OF - <(v, w, v') ∈ edges
G>]
  unfolding valid-unMultigraph-def
  by auto
moreover have valid-unMultigraph.connected (del-unEdge v w v' G)
  using <finite (edges G)> <finite (nodes G)> <valid-unMultigraph G>
  <valid-unMultigraph.connected G>
by (metis <(v, w, v') ∈ edges G> all-even valid-unMultigraph.del-unEdge-even-connectivity)
moreover have nodes(del-unEdge v w v' G) ≠ {}
  by (metis <v ∈ nodes G> del-UnEdge-node emptyE)
ultimately obtain n1 n2 ps where
  n1-n2:
  n1 ∈ nodes (del-unEdge v w v' G)
  n2 ∈ nodes (del-unEdge v w v' G)
  odd (degree n1 (del-unEdge v w v' G))
  odd (degree n2 (del-unEdge v w v' G))
  n1 ≠ n2
  and
  ps-eulerian:
  valid-unMultigraph.is-Eulerian-trail (del-unEdge v w v' G) n1 ps n2
  by (metis <num-of-odd-nodes (del-unEdge v w v' G) = 2> less.hyps(1))
have n1 = v ⇒ n2 = v' ⇒ valid-unMultigraph.is-Eulerian-circuit G v (ps@[v, w, v])
v
  using ps-eulerian
  by (metis <(v, w, v') ∈ edges G> delete-edge-sym <valid-unMultigraph G>
  valid-unMultigraph.corres valid-unMultigraph.eulerian-cons'
  valid-unMultigraph.is-Eulerian-circuit-def)
moreover have n1 = v' ⇒ n2 = v ⇒ ∃ ps. valid-unMultigraph.is-Eulerian-circuit
G v ps v
  by (metis <(v, w, v') ∈ edges G> <valid-unMultigraph G> ps-eulerian
  valid-unMultigraph.eulerian-cons valid-unMultigraph.is-Eulerian-circuit-def)
moreover have (n1 = v ∧ n2 = v') ∨ (n2 = v ∧ n1 = v')
  by (metis (mono-tags) all-even del-UnEdge-node insert-iff <finite (edges G)>
  <valid-unMultigraph G> n1-n2(1) n1-n2(2) n1-n2(3) n1-n2(4) n1-n2(5))
singletonE
  valid-unMultigraph.degree-frame)
  ultimately show ∃ ps. valid-unMultigraph.is-Eulerian-circuit G v ps v by
auto
  qed
qed
end

```

```

theory FriendshipTheory
  imports MoreGraph HOL-Number-Theory.Number-Theory
begin

```

10 Common steps

definition (in *valid-unSimpGraph*) *non-adj* :: 'v ⇒ 'v ⇒ bool **where**
non-adj v v' ≡ v ∈ V ∧ v' ∈ V ∧ v ≠ v' ∧ ¬adjacent v v'

lemma (in *valid-unSimpGraph*) *no-quad*:

assumes $\bigwedge v u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. \text{adjacent } v \ n \ \wedge \ \text{adjacent } u \ n$
shows $\neg (\exists v1 \ v2 \ v3 \ v4. v2 \neq v4 \ \wedge \ v1 \neq v3 \ \wedge \ \text{adjacent } v1 \ v2 \ \wedge \ \text{adjacent } v2 \ v3 \ \wedge$
adjacent v3 v4
 $\wedge \ \text{adjacent } v4 \ v1)$

proof

assume $\exists v1 \ v2 \ v3 \ v4. v2 \neq v4 \ \wedge \ v1 \neq v3 \ \wedge \ \text{adjacent } v1 \ v2 \ \wedge \ \text{adjacent } v2 \ v3 \ \wedge$
adjacent v3 v4 $\wedge \ \text{adjacent } v4 \ v1$

then obtain v1 v2 v3 v4 **where**

v2 ≠ v4 v1 ≠ v3 adjacent v1 v2 adjacent v2 v3 adjacent v3 v4 adjacent v4 v1

by auto

hence $\exists! n. \text{adjacent } v1 \ n \ \wedge \ \text{adjacent } v3 \ n$ **using** *assms*[of v1 v3] **by** auto

thus False

by (metis <adjacent v1 v2> <adjacent v2 v3> <adjacent v3 v4> <adjacent v4 v1>
<v2 ≠ v4>
adjacent-sym)

qed

lemma *even-card-set*:

assumes *finite* A **and** $\forall x \in A. f \ x \in A \ \wedge \ f \ x \neq x \ \wedge \ f \ (f \ x) = x$

shows *even*(card A) **using** *assms*

proof (*induct* card A *arbitrary*:A *rule*:less-induct)

case *less*

have A = {} \implies ?case **by** auto

moreover have A ≠ {} \implies ?case

proof –

assume A ≠ {}

then obtain x **where** x ∈ A **by** auto

hence f x ∈ A **and** f x ≠ x **by** (metis less.prem(2))+

obtain B **where** 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 $\forall x \in B. f \ x \in B \ \wedge \ f \ x \neq x \ \wedge \ f \ (f \ x) = x$

proof

fix y **assume** y ∈ B

hence y ∈ A **using** B **by** auto

hence f y ≠ y **and** f (f y) = y **by** (metis less.prem(2))+

moreover have f y ∈ B


```

proof (rule ccontr)
  assume  $f y \notin B$ 
  have  $f y \in A$  by (metis  $\langle y \in A \rangle$  less.prem2)
  hence  $f y \in \{x, f x\}$  by (metis  $B$  DiffI  $\langle f y \notin B \rangle$ )
  moreover have  $f y = x \implies \text{False}$ 
    by (metis  $B$  Diff-iff Diff-insert2  $\langle f (f y) = y \rangle$   $\langle y \in B \rangle$  singleton-iff)
  moreover have  $f y = f x \implies \text{False}$ 
    by (metis  $B$  Diff-iff  $\langle x \in A \rangle$   $\langle y \in B \rangle$  insertCI less.prem2)
  ultimately show  $\text{False}$  by auto
qed
  ultimately show  $f y \in B \wedge f y \neq y \wedge f (f y) = y$  by auto
qed
ultimately have  $\text{even} (\text{card } B)$  by (metis (full-types) less.hyps)
moreover have  $\{x, f x\} \subseteq A$  using  $\langle f x \in A \rangle$   $\langle x \in A \rangle$  by auto
moreover have  $\text{card } \{x, f x\} = 2$  using  $\langle f x \neq x \rangle$  by auto
ultimately show ?case using  $B$   $\langle \text{finite } A \rangle$  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:  $\bigwedge v u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. \text{adjacent } v n \wedge \text{adjacent } u n$ 
  and finite  $E$ 
  shows  $\forall v \in V. \text{even}(\text{degree } v G)$ 
proof
  fix  $v$  assume  $v \in V$ 
  obtain  $f$  where  $f: f = (\lambda n. (\text{SOME } v'. n \in V \longrightarrow n \neq v \longrightarrow \text{adjacent } n v' \wedge \text{adjacent } v v'))$  by auto
  have  $\bigwedge n. n \in V \longrightarrow n \neq v \longrightarrow (\exists v'. \text{adjacent } n v' \wedge \text{adjacent } v v')$ 
  proof (rule, rule)
    fix  $n$  assume  $n \in V$   $n \neq v$ 
    hence  $\exists! v'. \text{adjacent } n v' \wedge \text{adjacent } v v'$ 
    using friend-assm[of  $n v$ ]  $\langle v \in V \rangle$  unfolding non-adj-def by auto
    thus  $\exists v'. \text{adjacent } n v' \wedge \text{adjacent } v v'$  by auto
  qed
hence  $f\text{-ex}: \bigwedge n. (\exists v'. n \in V \longrightarrow n \neq v \longrightarrow \text{adjacent } n v' \wedge \text{adjacent } v v')$  by auto
have  $\forall x \in \{n. \text{adjacent } v n\}. f x \in \{n. \text{adjacent } v n\} \wedge f x \neq x \wedge f (f x) = x$ 
proof
  fix  $x$  assume  $x \in \{n. \text{adjacent } v n\}$ 
  hence  $\text{adjacent } v x$  by auto
  have  $f x \in \{n. \text{adjacent } v n\}$ 
  using someI-ex[OF  $f\text{-ex}$ , of  $x$ ]
  by (metis  $\langle \text{adjacent } v x \rangle$  adjacent-V(2) adjacent-no-loop  $f$  mem-Collect-eq)
  moreover have  $f x \neq x$ 
  using someI-ex[OF  $f\text{-ex}$ , of  $x$ ]
  by (metis  $\langle \text{adjacent } v x \rangle$  adjacent-V(2) adjacent-no-loop  $f$ )
  moreover have  $f (f x) = x$ 

```

proof (*rule ccontr*)
assume $f (f x) \neq x$
have $\text{adjacent } (f x) (f (f x))$
using $\text{someI-ex}[OF f\text{-ex, of } f x]$
by (*metis (full-types) adjacent-V(2) adjacent-no-loop calculation(1) f mem-Collect-eq*)
moreover have $\text{adjacent } (f (f x)) v$
using $\text{someI-ex}[OF f\text{-ex, of } f x]$ **by** (*metis adjacent-V(1) adjacent-sym calculation f*)
moreover have $\text{adjacent } x (f x)$
using $\text{someI-ex}[OF f\text{-ex, of } f x]$ **by** (*metis <adjacent v x> adjacent-V(2) adjacent-no-loop f*)
moreover have $v \neq f x$
by (*metis <f x ∈ {n. adjacent v n}> adjacent-no-loop mem-Collect-eq*)
ultimately show *False*
using $\text{no-quad}[OF \text{friend-assm}]$ **using** $\langle \text{adjacent } v x \rangle \langle f (f x) \neq x \rangle$
by *metis*
qed
ultimately show $f x \in \{n. \text{adjacent } v n\} \wedge f x \neq x \wedge f (f x) = x$ **by** *auto*
qed
moreover have $\text{finite } \{n. \text{adjacent } v n\}$ **by** (*metis adjacent-finite assms(2)*)
ultimately have $\text{even } (\text{card } \{n. \text{adjacent } v n\})$
using $\text{even-card-set}[of \{n. \text{adjacent } v n\} f]$ **by** *auto*
thus $\text{even } (\text{degree } v G)$ **by** (*metis assms(2) degree-adjacent*)
qed

lemma (*in valid-unSimpGraph*) *degree-two-windmill*:

assumes $\text{friend-assm}: \bigwedge v u. v \in V \implies u \in V \implies v \neq u \implies \exists ! n. \text{adjacent } v n \wedge \text{adjacent } u n$

and $\text{finite } E$ **and** $\text{card } V \geq 2$

shows $(\exists v \in V. \text{degree } v G = 2) \longleftrightarrow (\exists v. \forall n \in V. n \neq v \longrightarrow \text{adjacent } v n)$

proof

assume $\exists v \in V. \text{degree } v G = 2$

then obtain v **where** $\text{degree } v G = 2$ **by** *auto*

hence $\text{card } \{n. \text{adjacent } v n\} = 2$ **using** $\text{degree-adjacent}[OF \langle \text{finite } E \rangle, of v]$ **by** *auto*

then obtain $v1 v2$ **where** $v1 v2: \{n. \text{adjacent } v n\} = \{v1, v2\}$ **and** $v1 \neq v2$

proof –

obtain $v1 S$ **where** $\{n. \text{adjacent } v n\} = \text{insert } v1 S$ **and** $v1 \notin S$ **and** $\text{card } S = 1$

using $\langle \text{card } \{n. \text{adjacent } v n\} = 2 \rangle$ $\text{card-Suc-eq}[of \{n. \text{adjacent } v n\} 1]$ **by** *auto*

then obtain $v2$ **where** $S = \text{insert } v2 \{\}$

using $\text{card-Suc-eq}[of S 0]$ **by** *auto*

hence $\{n. \text{adjacent } v n\} = \{v1, v2\}$ **and** $v1 \neq v2$

using $\langle \{n. \text{adjacent } v n\} = \text{insert } v1 S \rangle \langle v1 \notin S \rangle$ **by** *auto*

thus *?thesis* **using** $\text{that}[of v1 v2]$ **by** *auto*

qed

have $\text{adjacent } v1 v2$

proof –
obtain n **where** $\text{adjacent } v \ n \ \text{adjacent } v1 \ n$ **using** $\text{friend-assm}[of \ v \ v1]$
by $(\text{metis } (\text{full-types}) \ \text{adjacent-V}(2) \ \text{adjacent-sym} \ \text{insertI1} \ \text{mem-Collect-eq} \ v1v2)$
hence $n \in \{n. \ \text{adjacent } v \ n\}$ **by** auto
moreover have $n \neq v1$ **by** $(\text{metis } \langle \text{adjacent } v1 \ n \rangle \ \text{adjacent-no-loop})$
ultimately have $n = v2$ **using** $v1v2$ **by** auto
thus $?thesis$ **by** $(\text{metis } \langle \text{adjacent } v1 \ n \rangle)$
qed
have $v1v2\text{-adj}:\forall x \in V. \ x \in \{n. \ \text{adjacent } v1 \ n\} \cup \{n. \ \text{adjacent } v2 \ n\}$
proof
fix x **assume** $x \in V$
have $x = v \implies x \in \{n. \ \text{adjacent } v1 \ n\} \cup \{n. \ \text{adjacent } v2 \ n\}$
by $(\text{metis } \text{Un-iff} \ \text{adjacent-sym} \ \text{insertI1} \ \text{mem-Collect-eq} \ v1v2)$
moreover have $x \neq v \implies x \in \{n. \ \text{adjacent } v1 \ n\} \cup \{n. \ \text{adjacent } v2 \ n\}$
proof –
assume $x \neq v$
then obtain y **where** $\text{adjacent } v \ y \ \text{adjacent } x \ y$
using $\text{friend-assm}[of \ v \ x]$
by $(\text{metis } \text{Collect-empty-eq} \ \langle x \in V \rangle \ \text{adjacent-V}(1) \ \text{all-not-in-conv} \ \text{insertCI} \ v1v2)$
hence $y = v1 \vee y = v2$ **using** $v1v2$ **by** auto
thus $x \in \{n. \ \text{adjacent } v1 \ n\} \cup \{n. \ \text{adjacent } v2 \ n\}$ **using** $\langle \text{adjacent } x \ y \rangle$
by $(\text{metis } \text{UnI1} \ \text{UnI2} \ \text{adjacent-sym} \ \text{mem-Collect-eq})$
qed
ultimately show $x \in \{n. \ \text{adjacent } v1 \ n\} \cup \{n. \ \text{adjacent } v2 \ n\}$ **by** auto
qed
have $\{n. \ \text{adjacent } v1 \ n\} - \{v2, v\} = \{\} \implies \exists v. \ \forall n \in V. \ n \neq v \longrightarrow \text{adjacent } v \ n$
proof $(\text{rule } \text{exI}[of \ - \ v2], \text{rule}, \text{rule})$
fix n **assume** $v1\text{-adj}:\{n. \ \text{adjacent } v1 \ n\} - \{v2, v\} = \{\}$ **and** $n \in V$ **and** $n \neq v2$
have $n \in \{n. \ \text{adjacent } v2 \ n\}$
proof $(\text{cases } n = v)$
case True
show $?thesis$ **by** $(\text{metis } \text{True} \ \text{adjacent-sym} \ \text{insertI1} \ \text{insert-commute} \ \text{mem-Collect-eq} \ v1v2)$
next
case False
have $n \notin \{n. \ \text{adjacent } v1 \ n\}$ **by** $(\text{metis } \text{DiffI} \ \text{False} \ \langle n \neq v2 \rangle \ \text{empty-iff} \ \text{insert-iff} \ v1\text{-adj})$
thus $?thesis$ **by** $(\text{metis } \text{Un-iff} \ \langle n \in V \rangle \ v1v2\text{-adj})$
qed
thus $\text{adjacent } v2 \ n$ **by** auto
qed
moreover have $\{n. \ \text{adjacent } v2 \ n\} - \{v1, v\} = \{\} \implies \exists v. \ \forall n \in V. \ n \neq v \longrightarrow \text{adjacent } v \ n$
proof $(\text{rule } \text{exI}[of \ - \ v1], \text{rule}, \text{rule})$
fix n **assume** $v2\text{-adj}:\{n. \ \text{adjacent } v2 \ n\} - \{v1, v\} = \{\}$ **and** $n \in V$ **and** $n \neq v1$

```

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  $\langle n \neq v1 \rangle$  empty-iff insert-iff v2-adj)
      thus ?thesis by (metis Un-iff  $\langle n \in V \rangle$  v1v2-adj)
      qed
      thus adjacent v1 n by auto
    qed
  moreover have  $\{n. \text{adjacent } v1 \ n\} - \{v2, v\} \neq \{\} \implies \{n. \text{adjacent } v2 \ n\} - \{v1, v\} \neq \{\}$ 
 $\implies$  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  $\langle a=b \rangle$  adjacent-sym by auto
    moreover have  $a \neq v$  by (metis DiffD2  $\langle a = b \rangle$  b doubleton-eq-iff insertI1)
    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-asm]
    using  $\langle v1 \neq v2 \rangle$  by auto
  qed
  moreover have  $a \neq b \implies$  False
  proof –
    assume  $a \neq b$ 
    moreover have  $a \in V$  using a by (metis DiffD1 adjacent-V(2) mem-Collect-eq)
    moreover have  $b \in V$  using b by (metis DiffD1 adjacent-V(2) mem-Collect-eq)
    ultimately obtain c where adjacent a c adjacent b c
    using friend-asm[of a b] by auto
    hence  $c \in \{n. \text{adjacent } v1 \ n\} \cup \{n. \text{adjacent } v2 \ n\}$ 
    by (metis (full-types) adjacent-V(2) v1v2-adj)
    moreover have  $c \in \{n. \text{adjacent } v1 \ n\} \implies$  False
    proof –
      assume  $c \in \{n. \text{adjacent } v1 \ n\}$ 
      hence adjacent v1 c by auto
      moreover have adjacent c b by (metis  $\langle \text{adjacent } b \ c \rangle$  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 $\neg c \neq v2$
 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 $c \in \{n. \text{adjacent } v2 \ n\} \implies \text{False}$
proof –
 assume $c \in \{n. \text{adjacent } v2 \ n\}$
 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 $\neg c \neq v1$
 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 b by (metis Diff-iff b mem-Collect-eq)
 ultimately have $b=v$ using friend-assm[of v1 v2]
 by (metis ‹v1 ≠ v2› adjacent-V(1))
 thus False using b by auto
 qed
moreover have a≠v2 by (metis DiffD2 a insertI1)
 ultimately show False using no-quad[OF friend-assm] by auto
 qed
 ultimately show False by auto
 qed
 ultimately show False by auto

qed
ultimately show $\exists v. \forall n \in V. n \neq v \longrightarrow \text{adjacent } v \ n$ **by auto**
next
assume $\exists v. \forall n \in V. n \neq v \longrightarrow \text{adjacent } v \ n$
then obtain v **where** $v: \forall n \in V. n \neq v \longrightarrow \text{adjacent } v \ n$ **by auto**
obtain $v1$ **where** $v1 \in V \ v1 \neq v$
proof (*cases* $v \in V$)
 case *False*
 have $V \neq \{\}$ **using** $\langle 2 \leq \text{card } V \rangle$ **by auto**
 then obtain $v1$ **where** $v1 \in V$ **by auto**
 thus *?thesis* **using** *False that*[*of* $v1$] **by auto**
next
 case *True*
 then obtain S **where** $V = \text{insert } v \ S \ v \notin S$
 using *mk-disjoint-insert*[*OF True*] **by auto**
 moreover have *finite* V **using** $\langle 2 \leq \text{card } V \rangle$
 by (*metis add-leE card.infinite not-one-le-zero numeral-Bit0 numeral-One*)
 ultimately have $1 \leq \text{card } S$
 using $\langle 2 \leq \text{card } V \rangle$ *card.insert*[*of* $S \ v$] *finite-insert*[*of* $v \ S$] **by auto**
 hence $S \neq \{\}$ **by auto**
 then obtain $v1$ **where** $v1 \in S$ **by auto**
 hence $v1 \neq v$ **using** $\langle v \notin S \rangle$ **by auto**
 thus *thesis* **using** *that*[*of* $v1$] $\langle v1 \in S \rangle \langle V = \text{insert } v \ S \rangle$ **by auto**
qed
hence $v \in V$ **using** v **by** (*metis adjacent-V(1)*)
then obtain $v2$ **where** *adjacent* $v1 \ v2$ *adjacent* $v \ v2$ **using** *friend-asm*[*of* $v \ v1$]

by (*metis* $\langle v1 \in V \rangle \langle v1 \neq v \rangle$)
have *degree* $v1 \ G \neq 2 \implies \text{False}$
proof –
 assume *degree* $v1 \ G \neq 2$
 hence *card* $\{n. \text{adjacent } v1 \ n\} \neq 2$ **by** (*metis assms(2) degree-adjacent*)
 have $\{v, v2\} \subseteq \{n. \text{adjacent } v1 \ n\}$
 by (*metis* $\langle \text{adjacent } v1 \ v2 \rangle \langle v1 \in V \rangle \langle v1 \neq v \rangle$ *adjacent-sym bot-least*
insert-subset
 mem-Collect-eq v)
 moreover have $v \neq v2$ **using** $\langle \text{adjacent } v \ v2 \rangle$ *adjacent-no-loop* **by auto**
 hence *card* $\{v, v2\} = 2$ **by auto**
 ultimately have *card* $\{n. \text{adjacent } v1 \ n\} \geq 2$
 using *adjacent-finite*[*OF* $\langle \text{finite } E \rangle$, *of* $v1$] **by** (*metis card-mono*)
 hence *card* $\{n. \text{adjacent } v1 \ n\} \geq 3$ **using** $\langle \text{card } \{n. \text{adjacent } v1 \ n\} \neq 2 \rangle$ **by**
auto
 then obtain $v3$ **where** $v3 \in \{n. \text{adjacent } v1 \ n\}$ **and** $v3 \notin \{v, v2\}$
 using $\langle \{v, v2\} \subseteq \{n. \text{adjacent } v1 \ n\} \rangle \langle \text{card } \{v, v2\} = 2 \rangle$
 by (*metis* $\langle \text{card } \{n. \text{adjacent } v1 \ n\} \neq 2 \rangle$ *subsetI subset-antisym*)
 hence *adjacent* $v1 \ v3$ **by auto**
 moreover have *adjacent* $v3 \ v$ **using** v
 by (*metis* $\langle v3 \notin \{v, v2\} \rangle$ *adjacent-V(2) adjacent-sym calculation insertCI*)
 moreover have *adjacent* $v \ v2$ **using** $\langle \text{adjacent } v \ v2 \rangle$.

moreover have $\text{adjacent } v2 \ v1$ **using** $\langle \text{adjacent } v1 \ v2 \rangle$ **adjacent-sym** **by auto**
moreover have $v1 \neq v$ **using** $\langle v1 \neq v \rangle$.
moreover have $v3 \neq v2$ **by** $(\text{metis } \langle v3 \notin \{v, v2\} \rangle \text{insert-subset subset-insertI})$
ultimately show False **using** $\text{no-quad}[OF \ \text{friend-assm}]$ **by auto**
qed
thus $\exists v \in V. \text{degree } v \ G = 2$ **using** $\langle v1 \in V \rangle$ **by auto**
qed

lemma (in valid-unSimpGraph) *regular*:
assumes $\text{friend-assm} : \bigwedge v \ u. v \in V \implies u \in V \implies v \neq u \implies \exists ! n. \text{adjacent } v \ n \wedge \text{adjacent } u \ n$
and $\text{finite } E$ **and** $\text{finite } V$ **and** $\neg(\exists v \in V. \text{degree } v \ G = 2)$
shows $\exists k. \forall v \in V. \text{degree } v \ G = k$
proof –
{ **fix** $v \ u$ **assume** $\text{non-adj } v \ u$
obtain $v\text{-adj}$ **where** $v\text{-adj} : v\text{-adj} = \{n. \text{adjacent } v \ n\}$ **by auto**
obtain $u\text{-adj}$ **where** $u\text{-adj} : u\text{-adj} = \{n. \text{adjacent } u \ n\}$ **by auto**
obtain f **where** $f : f = (\lambda n. (\text{SOME } v'. n \in V \longrightarrow n \neq u \longrightarrow \text{adjacent } n \ v' \wedge \text{adjacent } u \ v'))$ **by auto**
have $\bigwedge n. n \in V \longrightarrow n \neq u \longrightarrow (\exists v'. \text{adjacent } n \ v' \wedge \text{adjacent } u \ v')$
proof (rule,rule)
fix n **assume** $n \in V \ n \neq u$
hence $\exists ! v'. \text{adjacent } n \ v' \wedge \text{adjacent } u \ v'$
using $\text{friend-assm}[of \ n \ u]$ $\langle \text{non-adj } v \ u \rangle$ **unfolding** non-adj-def **by auto**
thus $\exists v'. \text{adjacent } n \ v' \wedge \text{adjacent } u \ v'$ **by auto**
qed
hence $f\text{-ex} : \bigwedge n. (\exists v'. n \in V \longrightarrow n \neq u \longrightarrow \text{adjacent } n \ v' \wedge \text{adjacent } u \ v')$ **by auto**
obtain $v\text{-adj-}u$ **where** $v\text{-adj-}u : v\text{-adj-}u = f \ 'v\text{-adj}$ **by auto**
have $\text{finite } u\text{-adj}$ **using** $u\text{-adj}$ $\text{adjacent-finite}[OF \ \langle \text{finite } E \rangle]$ **by auto**
have $\text{finite } v\text{-adj}$ **using** $v\text{-adj}$ $\text{adjacent-finite}[OF \ \langle \text{finite } E \rangle]$ **by auto**
hence $\text{finite } v\text{-adj-}u$ **using** $v\text{-adj-}u$ $\text{adjacent-finite}[OF \ \langle \text{finite } E \rangle]$ **by auto**
have $\text{inj-on } f \ v\text{-adj}$ **unfolding** inj-on-def
proof (rule ccontr)
assume $\neg (\forall x \in v\text{-adj}. \forall y \in v\text{-adj}. f \ x = f \ y \longrightarrow x = y)$
then obtain $x \ y$ **where** $x \in v\text{-adj} \ y \in v\text{-adj} \ f \ x = f \ y \ x \neq y$ **by auto**
have $x \in V$ **by** $(\text{metis } \langle x \in v\text{-adj} \rangle \text{adjacent-V}(2) \text{mem-Collect-eq } v\text{-adj})$
moreover have $x \neq u$ **by** $(\text{metis } \langle \text{non-adj } v \ u \rangle \langle x \in v\text{-adj} \rangle \text{mem-Collect-eq } v\text{-adj})$
ultimately have $\text{adjacent } (f \ x) \ u$ **and** $\text{adjacent } x \ (f \ x)$
using $\text{someI-ex}[OF \ f\text{-ex}[of \ x]]$ adjacent-sym **by** $(\text{metis } f) +$
hence $f \ x \neq v$ **by** $(\text{metis } \langle \text{non-adj } v \ u \rangle \text{non-adj-def})$
have $y \in V$ **by** $(\text{metis } \langle y \in v\text{-adj} \rangle \text{adjacent-V}(2) \text{mem-Collect-eq } v\text{-adj})$
moreover have $y \neq u$ **by** $(\text{metis } \langle \text{non-adj } v \ u \rangle \langle y \in v\text{-adj} \rangle \text{mem-Collect-eq } v\text{-adj})$
ultimately have $\text{adjacent } y \ (f \ y)$ **using** $\text{someI-ex}[OF \ f\text{-ex}[of \ y]]$ **by** $(\text{metis } f)$
hence $x \neq y \wedge v \neq f \ x \wedge \text{adjacent } v \ x \wedge \text{adjacent } x \ (f \ x) \wedge \text{adjacent } (f \ x) \ y \wedge \text{adjacent } y \ v$

using $\langle x \in v\text{-adj} \rangle \langle y \in v\text{-adj} \rangle \langle f x = f y \rangle \langle x \neq y \rangle \langle \text{adjacent } x (f x) \rangle v\text{-adj}$
adjacent-sym $\langle f x \neq v \rangle$
by auto
thus *False* **using** *no-quad*[*OF friend-assm*] **by auto**
qed
then have $\text{card } v\text{-adj} = \text{card } v\text{-adj-}u$ **by** (*metis card-image v-adj-u*)
moreover have $v\text{-adj-}u \subseteq u\text{-adj}$
proof
fix x **assume** $x \in v\text{-adj-}u$
then obtain y **where** $y \in v\text{-adj}$
and $x = (\text{SOME } v'. y \in V \longrightarrow y \neq u \longrightarrow \text{adjacent } y v' \wedge \text{adjacent } u v')$
using *f image-def v-adj-u* **by auto**
hence $y \in V \longrightarrow y \neq u \longrightarrow \text{adjacent } y x \wedge \text{adjacent } u x$ **using** *someI-ex*[*OF f-ex*[*of y*]]
by auto
moreover have $y \in V$ **by** (*metis* $\langle y \in v\text{-adj} \rangle$ *adjacent-V*(2) *mem-Collect-eq v-adj*)
moreover have $y \neq u$ **by** (*metis* $\langle \text{non-adj } v u \rangle \langle y \in v\text{-adj} \rangle$ *mem-Collect-eq non-adj-def v-adj*)
ultimately have $\text{adjacent } u x$ **by auto**
thus $x \in u\text{-adj}$ **unfolding** *u-adj* **by auto**
qed
moreover have $\text{card } v\text{-adj} = \text{degree } v G$ **using** *degree-adjacent*[*OF* $\langle \text{finite } E \rangle$, *of v*] *v-adj* **by auto**
moreover have $\text{card } u\text{-adj} = \text{degree } u G$ **using** *degree-adjacent*[*OF* $\langle \text{finite } E \rangle$, *of u*] *u-adj* **by auto**
ultimately have $\text{degree } v G \leq \text{degree } u G$ **using** $\langle \text{finite } u\text{-adj} \rangle$
by (*metis* $\langle \text{inj-on } f v\text{-adj} \rangle$ *card-inj-on-le v-adj-u*) }
hence $\text{non-adj-degree} : \bigwedge v u. \text{non-adj } v u \implies \text{degree } v G = \text{degree } u G$
by (*metis adjacent-sym antisym non-adj-def*)
have $\text{card } V = 3 \implies ?thesis$
proof
assume $\text{card } V = 3$
then obtain $v1 v2 v3$ **where** $V = \{v1, v2, v3\}$ $v1 \neq v2$ $v2 \neq v3$ $v1 \neq v3$
proof –
obtain $v1 S1$ **where** $VS1 : V = \text{insert } v1 S1$ **and** $v1 \notin S1$ **and** $\text{card } S1 = 2$
using *card-Suc-eq*[*of V 2*] $\langle \text{card } V = 3 \rangle$ **by auto**
then obtain $v2 S2$ **where** $S1S2 : S1 = \text{insert } v2 S2$ **and** $v2 \notin S2$ **and** $\text{card } S2 = 1$
using *card-Suc-eq*[*of S1 1*] **by auto**
then obtain $v3$ **where** $S2 = \{v3\}$
using *card-Suc-eq*[*of S2 0*] **by auto**
hence $V = \{v1, v2, v3\}$ **using** $VS1 S1S2$ **by auto**
moreover have $v1 \neq v2$ $v2 \neq v3$ $v1 \neq v3$ **using** $VS1 S1S2 \langle v1 \notin S1 \rangle \langle v2 \notin S2 \rangle \langle S2 = \{v3\} \rangle$ **by auto**
ultimately show *?thesis* **using that** **by auto**
qed
obtain n **where** $\text{adjacent } v1 n$ $\text{adjacent } v2 n$

using *friend-assm*[of $v1\ v2$] **by** (*metis* $\langle V = \{v1, v2, v3\} \rangle \langle v1 \neq v2 \rangle$ *insertI1*
insertI2)
moreover **hence** $n=v3$
using $\langle V = \{v1, v2, v3\} \rangle$ *adjacent-V(2)* *adjacent-no-loop*
by (*metis* (*mono-tags*) *empty-iff* *insertE*)
moreover **obtain** n' **where** *adjacent* $v2\ n'$ *adjacent* $v3\ n'$
using *friend-assm*[of $v2\ v3$] **by** (*metis* $\langle V = \{v1, v2, v3\} \rangle \langle v2 \neq v3 \rangle$ *insertI1*
insertI2)
moreover **hence** $n'=v1$
using $\langle V = \{v1, v2, v3\} \rangle$ *adjacent-V(2)* *adjacent-no-loop*
by (*metis* (*mono-tags*) *empty-iff* *insertE*)
ultimately **have** *adjacent* $v1\ v2$ **and** *adjacent* $v2\ v3$ **and** *adjacent* $v3\ v1$
using *adjacent-sym* **by** *auto*
have *degree* $v1\ G=2$
proof –
have $v2 \in \{n. \text{adjacent } v1\ n\}$ **and** $v3 \in \{n. \text{adjacent } v1\ n\}$ **and** $v1 \notin \{n.$
adjacent $v1\ n\}$
using $\langle \text{adjacent } v1\ v2 \rangle \langle \text{adjacent } v3\ v1 \rangle$ *adjacent-sym*
by (*auto,metis adjacent-no-loop*)
hence $\{n. \text{adjacent } v1\ n\} = \{v2, v3\}$ **using** $\langle V = \{v1, v2, v3\} \rangle$ **by** *auto*
thus *?thesis* **using** *degree-adjacent*[*OF* $\langle \text{finite } E \rangle, \text{of } v1$] $\langle v2 \neq v3 \rangle$ **by** *auto*
qed
moreover **have** *degree* $v2\ G=2$
proof –
have $v1 \in \{n. \text{adjacent } v2\ n\}$ **and** $v3 \in \{n. \text{adjacent } v2\ n\}$ **and** $v2 \notin \{n.$
adjacent $v2\ n\}$
using $\langle \text{adjacent } v1\ v2 \rangle \langle \text{adjacent } v2\ v3 \rangle$ *adjacent-sym*
by (*auto,metis adjacent-no-loop*)
hence $\{n. \text{adjacent } v2\ n\} = \{v1, v3\}$ **using** $\langle V = \{v1, v2, v3\} \rangle$ **by** *force*
thus *?thesis* **using** *degree-adjacent*[*OF* $\langle \text{finite } E \rangle, \text{of } v2$] $\langle v1 \neq v3 \rangle$ **by** *auto*

qed
moreover **have** *degree* $v3\ G=2$
proof –
have $v1 \in \{n. \text{adjacent } v3\ n\}$ **and** $v2 \in \{n. \text{adjacent } v3\ n\}$ **and** $v3 \notin \{n.$
adjacent $v3\ n\}$
using $\langle \text{adjacent } v3\ v1 \rangle \langle \text{adjacent } v2\ v3 \rangle$ *adjacent-sym*
by (*auto,metis adjacent-no-loop*)
hence $\{n. \text{adjacent } v3\ n\} = \{v1, v2\}$ **using** $\langle V = \{v1, v2, v3\} \rangle$ **by** *force*
thus *?thesis* **using** *degree-adjacent*[*OF* $\langle \text{finite } E \rangle, \text{of } v3$] $\langle v1 \neq v2 \rangle$ **by** *auto*
qed
ultimately **show** $\forall v \in V. \text{degree } v\ G = 2$ **using** $\langle V = \{v1, v2, v3\} \rangle$ **by** *auto*
qed
moreover **have** *card* $V=2 \implies \text{False}$
proof –
assume *card* $V=2$
obtain $v1\ v2$ **where** $V = \{v1, v2\}$ $v1 \neq v2$
proof –
obtain $v1\ S1$ **where** $VS1: V = \text{insert } v1\ S1$ **and** $v1 \notin S1$ **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
  hence degree v1 G=0 unfolding degree-def by auto
  thus ∀v∈V. degree v G =0 using ⟨V={v1}⟩ by auto
qed
moreover have card V=0 ⇒ ?thesis
proof -
  assume card V=0
  hence V={} using ⟨finite V⟩ by auto
  thus ?thesis by auto
qed
moreover have card V ≥4 ⇒ ¬(∃v u. non-adj v u) ⇒ False
proof -
  assume ¬(∃v u. non-adj v u) card V ≥4
  hence non-non-adj: ∧v u. v∈V ∨ u∈V ∨ v=u ∨ adjacent v u unfolding
non-adj-def by auto
  obtain v1 v2 v3 v4 where v1∈V v2∈V v3∈V v4∈V v1≠v2 v1≠v3 v1≠v4
    v2≠v3 v2≠v4 v3≠v4
  proof -
    obtain v1 B1 where V = insert v1 B1 v1 ∉ B1 card B1 ≥3
      using ⟨card V ≥4⟩ card-le-Suc-iff[of 3 V] by auto
    then obtain v2 B2 where B1 = insert v2 B2 v2 ∉ B2 card B2 ≥2
      using card-le-Suc-iff[of 2 B1] by auto
    then obtain v3 B3 where B2 = insert v3 B3 v3 ∉ B3 card B3 ≥1
      using card-le-Suc-iff[of 1 B2] by auto
    then obtain v4 B4 where B3 = insert v4 B4 v4 ∉ B4
```

using *card-le-Suc-iff*[of 0 B3] **by** *auto*
have $v1 \in V$ **by** (*metis* $\langle V = \text{insert } v1 \ B1 \rangle$ *insert-subset order-refl*)
moreover have $v2 \in V$
by (*metis* $\langle B1 = \text{insert } v2 \ B2 \rangle$ $\langle V = \text{insert } v1 \ B1 \rangle$ *insert-subset subset-insertI*)
moreover have $v3 \in V$
by (*metis* $\langle B1 = \text{insert } v2 \ B2 \rangle$ $\langle B2 = \text{insert } v3 \ B3 \rangle$ $\langle V = \text{insert } v1 \ B1 \rangle$ *insert-iff*)
moreover have $v4 \in V$
by (*metis* $\langle B1 = \text{insert } v2 \ B2 \rangle$ $\langle B2 = \text{insert } v3 \ B3 \rangle$ $\langle B3 = \text{insert } v4 \ B4 \rangle$ $\langle V = \text{insert } v1 \ B1 \rangle$ *insert-iff*)
moreover have $v1 \neq v2$
by (*metis* (*full-types*) $\langle B1 = \text{insert } v2 \ B2 \rangle$ $\langle v1 \notin B1 \rangle$ *insertI1*)
moreover have $v1 \neq v3$
by (*metis* $\langle B1 = \text{insert } v2 \ B2 \rangle$ $\langle B2 = \text{insert } v3 \ B3 \rangle$ $\langle v1 \notin B1 \rangle$ *insert-iff*)
moreover have $v1 \neq v4$
by (*metis* $\langle B1 = \text{insert } v2 \ B2 \rangle$ $\langle B2 = \text{insert } v3 \ B3 \rangle$ $\langle B3 = \text{insert } v4 \ B4 \rangle$ $\langle v1 \notin B1 \rangle$ *insert-iff*)
moreover have $v2 \neq v3$
by (*metis* (*full-types*) $\langle B2 = \text{insert } v3 \ B3 \rangle$ $\langle v2 \notin B2 \rangle$ *insertI1*)
moreover have $v2 \neq v4$
by (*metis* $\langle B2 = \text{insert } v3 \ B3 \rangle$ $\langle B3 = \text{insert } v4 \ B4 \rangle$ $\langle v2 \notin B2 \rangle$ *insert-iff*)
moreover have $v3 \neq v4$
by (*metis* (*full-types*) $\langle B3 = \text{insert } v4 \ B4 \rangle$ $\langle v3 \notin B3 \rangle$ *insertI1*)
ultimately show *?thesis using that by auto*
qed
hence *adjacent* $v1 \ v2$ **using** *non-non-adj* **by** *auto*
moreover have *adjacent* $v2 \ v3$ **using** *non-non-adj* **by** (*metis* $\langle v2 \in V \rangle$ $\langle v2 \neq v3 \rangle$ $\langle v3 \in V \rangle$)
moreover have *adjacent* $v3 \ v4$ **using** *non-non-adj* **by** (*metis* $\langle v3 \in V \rangle$ $\langle v3 \neq v4 \rangle$ $\langle v4 \in V \rangle$)
moreover have *adjacent* $v4 \ v1$ **using** *non-non-adj* **by** (*metis* $\langle v1 \in V \rangle$ $\langle v1 \neq v4 \rangle$ $\langle v4 \in V \rangle$)
ultimately show *False using no-quad*[OF *friend-assm*]
by (*metis* $\langle v1 \neq v3 \rangle$ $\langle v2 \neq v4 \rangle$)
qed
moreover have *card* $V \geq 4 \implies (\exists v \ u. \text{non-adj } v \ u) \implies ?thesis$
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*: $\forall n. \text{adjacent } v \ n \wedge \text{adjacent } u \ n \longrightarrow n=w$
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* $\langle \text{non-adj } v \ u \rangle$ **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* $\langle \text{non-adj } v \ u \rangle$ *non-adj-degree*)
moreover have $\neg \text{non-adj } u \ n \implies \neg \text{non-adj } v \ n \implies \text{degree } n \ G =$
degree } v \ G
by (*metis* $\langle n \in V \rangle \langle n \neq w \rangle \langle \text{non-adj } v \ u \rangle$ *non-adj-def 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 $\neg(\exists v. \forall n \in V. n \neq v \longrightarrow \text{adjacent } v \ n)$
using $\langle \text{card } V \geq 4 \rangle$ *degree-two-windmill* *assms(2)* *assms(4)* *friend-assm*
by *auto*
ultimately obtain $w1$ **where** $w1 \in V \ w1 \neq w \ \text{non-adj } w \ w1$
by (*metis* $\langle n \in V \rangle$ *non-adj-def*)
have $w1=v \implies \text{degree } n \ G = \text{degree } v \ G$
by (*metis* $\langle n = w \rangle \langle \text{non-adj } w \ w1 \rangle$ *non-adj-degree*)
moreover have $w1=u \implies \text{degree } n \ G = \text{degree } v \ G$
by (*metis* $\langle \text{adjacent } u \ w \rangle \langle \text{non-adj } w \ w1 \rangle$ *adjacent-sym non-adj-def*)
moreover have $w1 \neq u \implies w1 \neq v \implies \text{degree } n \ G = \text{degree } v \ G$
by (*metis* $\langle n = w \rangle \langle \text{non-adj } v \ u \rangle \langle \text{non-adj } w \ w1 \rangle$ *non-adj-def*
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 *?thesis* **by** *auto*
qed
ultimately show *?thesis* **by** *force*
qed

11 Exclusive steps for combinatorial proofs

fun (**in** *valid-unSimpGraph*) *adj-path*:: $'v \Rightarrow 'v \ \text{list} \Rightarrow \text{bool}$ **where**
adj-path $v \ [] = (v \in V)$
 $| \text{adj-path } v \ (u \# \text{us}) = (\text{adjacent } v \ u \wedge \text{adj-path } u \ \text{us})$

lemma (**in** *valid-unSimpGraph*) *adj-path-butlast*:
 $\text{adj-path } v \ ps \implies \text{adj-path } v \ (\text{butlast } ps)$
by (*induct ps arbitrary:v,auto*)

lemma (**in** *valid-unSimpGraph*) *adj-path-V*:
 $\text{adj-path } v \ ps \implies \text{set } ps \subseteq V$

by (*induct ps arbitrary:v, auto*)

lemma (*in valid-unSimpGraph*) *adj-path-V'*:

adj-path v ps $\implies v \in V$

by (*induct ps arbitrary:v, auto*)

lemma (*in valid-unSimpGraph*) *adj-path-app*:

adj-path v ps $\implies ps \neq [] \implies adjacent (last ps) u \implies adj-path v (ps@[u])$

proof (*induct ps arbitrary:v*)

case *Nil*

thus *?case by auto*

next

case (*Cons x xs*)

thus *?case by (cases xs, auto)*

qed

lemma (*in valid-unSimpGraph*) *adj-path-app'*:

adj-path v (ps @ [q]) $\implies ps \neq [] \implies adjacent (last ps) q$

proof (*induct ps arbitrary:v*)

case *Nil*

thus *?case by auto*

next

case (*Cons x xs*)

thus *?case by (cases xs, auto)*

qed

lemma *card-partition'*:

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

$\forall v1 v2. v1 \neq v2 \longrightarrow \{n. R v1 n\} \cap \{n. R v2 n\} = \{\}$

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

proof –

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

proof –

fix *C* **assume** $C \in (\lambda x. \{n. R x n\}) ' A$

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

imageE)

qed

moreover **have** $\bigwedge C1 C2. C1 \in (\lambda x. \{n. R x n\}) ' A \implies C2 \in (\lambda x. \{n. R x n\}) ' A$
 $C1 \neq C2 \implies C1 \cap C2 = \{\}$

$\implies C1 \cap C2 = \{\}$

proof –

fix *C1 C2* **assume** $C1 \in (\lambda x. \{n. R x n\}) ' A \ C2 \in (\lambda x. \{n. R x n\}) ' A$
 $C1 \neq C2$

obtain *v1* **where** $v1 \in A \ C1 = \{n. R v1 n\}$ **by** (*metis $\langle C1 \in (\lambda x. \{n. R x n\}) ' A \rangle \text{imageE}$*)

obtain *v2* **where** $v2 \in A \ C2 = \{n. R v2 n\}$ **by** (*metis $\langle C2 \in (\lambda x. \{n. R x n\}) ' A \rangle \text{imageE}$*)

have $v1 \neq v2$ **by** (*metis $\langle C1 = \{n. R v1 n\} \langle C1 \neq C2 \rangle \langle C2 = \{n. R v2 n\} \rangle$*)

thus $C1 \cap C2 = \{\}$ **by** (*metis* $\langle C1 = \{n. R v1 n\} \langle C2 = \{n. R v2 n\} \rangle$
assms(4))
qed
moreover have $\bigcup((\lambda x. \{n. R x n\}) ' A) = (\bigcup x \in A. \{n. R x n\})$ **by** *auto*
moreover have *finite* $((\lambda x. \{n. R x n\}) ' A)$ **by** (*metis* *assms(3)* *finite-imageI*)
moreover have *finite* $(\bigcup((\lambda x. \{n. R x n\}) ' A))$ **by** (*metis* (*full-types*) *assms(1)*)

assms(2) *assms(3)* *card-eq-0-iff* *finite-UN-I* *less-nat-zero-code*)
moreover have $\text{card } A = \text{card } ((\lambda x. \{n. R x n\}) ' A)$
proof –
have *inj-on* $(\lambda x. \{n. R x n\}) A$ **unfolding** *inj-on-def*
using $\langle \forall v1 v2. v1 \neq v2 \longrightarrow \{n. R v1 n\} \cap \{n. R v2 n\} = \{\} \rangle$
by (*metis* *assms(1)* *assms(2)* *card.empty* *inf.idem* *less-le*)
thus *?thesis* **by** (*metis* *card-image*)
qed
ultimately show *?thesis* **using** *card-partition*[*of* $(\lambda x. \{n. R x n\}) ' A$] **by** *auto*
qed

lemma (*in* *valid-unSimpGraph*) *path-count*:
assumes $k\text{-adj} : \bigwedge v. v \in V \implies \text{card } \{n. \text{adjacent } v n\} = k$ **and** $v \in V$ **and** *finite*
 V **and** $k > 0$
shows $\text{card } \{ps. \text{length } ps = l \wedge \text{adj-path } v ps\} = k^l$
proof (*induct* l *rule:nat.induct*)
case *zero*
have $\{ps. \text{length } ps = 0 \wedge \text{adj-path } v ps\} = \{\}\}$ **using** $\langle v \in V \rangle$ **by** *auto*
thus *?case* **by** *auto*
next
case (*Suc* n)
obtain *ext* **where** $\text{ext} = (\lambda ps ps'. ps' \neq [] \wedge (\text{butlast } ps' = ps) \wedge \text{adj-path } v ps')$
by *auto*
have $\forall ps \in \{ps. \text{length } ps = n \wedge \text{adj-path } v ps\}. \text{card } \{ps'. \text{ext } ps ps'\} = k$
proof
fix ps **assume** $ps \in \{ps. \text{length } ps = n \wedge \text{adj-path } v ps\}$
hence $\text{adj-path } v ps$ **and** $\text{length } ps = n$ **by** *auto*
obtain qs **where** $qs : qs = \{n. \text{if } ps = [] \text{ then } \text{adjacent } v n \text{ else } \text{adjacent } (\text{last } ps) n\}$ **by** *auto*
hence $\text{card } qs = k$
proof (*cases* $ps = []$)
case *True*
thus *?thesis* **using** qs *k-adj*[*OF* $\langle v \in V \rangle$] **by** *auto*
next
case *False*
have $\text{last } ps \in V$ **using** *adj-path-V* **by** (*metis* *False* $\langle \text{adj-path } v ps \rangle$
last-in-set *subsetD*)
thus *?thesis* **using** *k-adj*[*of* $\text{last } ps$] *False* qs **by** *auto*
qed
obtain *app* **where** $\text{app} : \text{app} = (\lambda q. ps @ [q])$ **by** *auto*
have $\text{app } ' qs = \{ps'. \text{ext } ps ps'\}$
proof –

```

have  $\bigwedge xs. xs \in \text{app } 'qs \implies xs \in \{ps'. \text{ext } ps \ ps'\}$ 
proof (rule, cases ps=[])
  case True
    fix xs assume xs  $\in$  app 'qs
    then obtain q where q  $\in$  qs app q=xs by (metis imageE)
    hence adjacent v q and xs=ps@[q] using qs app True by auto
    hence adj-path v xs
      by (metis True adj-path.simps(1) adj-path.simps(2) adjacent-V(2))
append-Nil)
    moreover have butlast xs = ps using  $\langle xs=ps@[q] \rangle$  by auto
    ultimately show ext ps xs using ext  $\langle xs=ps@[q] \rangle$  by auto
  next
    case False
    fix xs assume xs  $\in$  app 'qs
    then obtain q where q  $\in$  qs app q=xs by (metis imageE)
    hence adjacent (last ps) q using qs app False by auto
    hence adj-path v (ps@[q]) using  $\langle \text{adj-path } v \ ps \rangle$  False adj-path-app by
auto
      hence adj-path v xs by (metis  $\langle \text{app } q = xs \rangle$  app)
    moreover have butlast xs=ps by (metis  $\langle \text{app } q = xs \rangle$  app butlast-snoc)
    ultimately show ext ps xs by (metis False butlast.simps(1) ext)
  qed
moreover have  $\bigwedge xs. xs \in \{ps'. \text{ext } ps \ ps'\} \implies xs \in \text{app } 'qs$ 
proof (cases ps=[])
  case True
    hence qs = {n. adjacent v n} using qs by auto
    fix xs assume xs  $\in$  {ps'. ext ps ps'}
    hence xs  $\neq$  [] and (butlast xs=ps) and adj-path v xs using ext by auto
    thus xs  $\in$  app 'qs
      using True app  $\langle qs = \{n. \text{adjacent } v \ n\} \rangle$ 
      by (metis adj-path.simps(2) append-butlast-last-id append-self-conv2)
image-iff)
    mem-Collect-eq)
  next
    case False
    fix xs assume xs  $\in$  {ps'. ext ps ps'}
    hence xs  $\neq$  [] and (butlast xs=ps) and adj-path v xs using ext by auto
    then obtain q where xs=ps@[q] by (metis append-butlast-last-id)
    hence adjacent (last ps) q using  $\langle \text{adj-path } v \ xs \rangle$  False adj-path-app' by
auto
      thus xs  $\in$  app 'qs using qs
      by (metis (lifting, full-types) False  $\langle xs = ps \ @ \ [q] \rangle$  app imageI)
mem-Collect-eq)
  qed
  ultimately show ?thesis by auto
qed
moreover have inj-on app qs using app unfolding inj-on-def by auto
ultimately show card {ps'. ext ps ps'}=k by (metis  $\langle \text{card } qs = k \rangle$  card-image)
qed

```

moreover have $\forall ps1\ ps2. ps1 \neq ps2 \longrightarrow \{n. ext\ ps1\ n\} \cap \{n. ext\ ps2\ n\} = \{\}$
using *ext* **by** *auto*
moreover have *finite* $\{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}$
using *Suc.hyps* *assms* **by** (*auto* *intro*: *card-ge-0-finite*)
ultimately have $card\ (\bigcup v \in \{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}. \{n. ext\ v\ n\})$
 $= k * card\ \{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}$
using *card-partition* [of $\{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}$ *ext* *k*] $\langle k > 0 \rangle$ **by**
auto
moreover have $\{ps. length\ ps = n + 1 \wedge adj\text{-}path\ v\ ps\}$
 $= (\bigcup ps \in \{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}. \{ps'. ext\ ps\ ps'\})$
proof –
have $\bigwedge xs. xs \in \{ps. length\ ps = n + 1 \wedge adj\text{-}path\ v\ ps\} \implies$
 $xs \in (\bigcup ps \in \{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}. \{ps'. ext\ ps\ ps'\})$
proof –
fix *xs* **assume** $xs \in \{ps. length\ ps = n + 1 \wedge adj\text{-}path\ v\ ps\}$
hence *length* *xs* = *n + 1* **and** *adj-path* *v* *xs* **by** *auto*
hence *butlast* *xs* $\in \{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}$
using *adj-path-butlast* *length-butlast* *mem-Collect-eq* **by** *auto*
thus $xs \in (\bigcup ps \in \{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}. \{ps'. ext\ ps\ ps'\})$
using $\langle adj\text{-}path\ v\ xs \rangle \langle length\ xs = n + 1 \rangle$ *UN-iff* *ext* *length-greater-0-conv*

mem-Collect-eq
by *auto*
qed
moreover have $\bigwedge xs. xs \in (\bigcup ps \in \{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}. \{ps'.$
ext *ps* *ps'\}) \implies
 $xs \in \{ps. length\ ps = n + 1 \wedge adj\text{-}path\ v\ ps\}$
proof –
fix *xs* **assume** $xs \in (\bigcup ps \in \{ps. length\ ps = n \wedge adj\text{-}path\ v\ ps\}. \{ps'. ext\ ps$
*ps'\})
then obtain *ys* **where** *length* *ys* = *n* *adj-path* *v* *ys* *ext* *ys* *xs* **by** *auto*
hence *length* *xs* = *n + 1* **using** *ext* **by** *auto*
thus $xs \in \{ps. length\ ps = n + 1 \wedge adj\text{-}path\ v\ ps\}$
by (*metis* (*lifting*, *full-types*) $\langle ext\ ys\ xs \rangle$ *ext* *mem-Collect-eq*)
qed
ultimately show *?thesis* **by** *fast*
qed
ultimately show $card\ \{ps. length\ ps = (Suc\ n) \wedge adj\text{-}path\ v\ ps\} = k \wedge (Suc\ n)$
using *Suc.hyps* **by** *auto*
qed**

lemma (in *valid-unSimpGraph*) *total-v-num*:

assumes *friend-assm*: $\bigwedge v\ u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. adjacent\ v\ n \wedge adjacent\ u\ n$

and *finite* *E* **and** *finite* *V* **and** $V \neq \{\}$ **and** $\forall v \in V. degree\ v\ G = k$ **and** $k > 0$

shows $card\ V = k * k - k + 1$

proof –

have $k\text{-}adj: \bigwedge v. v \in V \implies card\ (\{n. adjacent\ v\ n\}) = k$ **by** (*metis* *assms*(2) *assms*(5) *degree-adjacent*)


```

obtain  $v$  where  $v \in V$  using  $\langle V \neq \{\} \rangle$  by auto
obtain  $l2\text{-eq-}v$  where  $l2\text{-eq-}v: l2\text{-eq-}v = \{ps. \text{length } ps = 2 \wedge \text{adj-path } v \text{ } ps \wedge \text{last } ps = v\}$  by auto
have  $\text{card } l2\text{-eq-}v = k$ 
proof –
  obtain  $hds$  where  $hds: hds = \text{hd } l2\text{-eq-}v$  by auto
  moreover have  $hds = \{n. \text{adjacent } v \text{ } n\}$ 
  proof –
    have  $\bigwedge x. x \in hds \implies x \in \{n. \text{adjacent } v \text{ } n\}$ 
    proof
      fix  $x$  assume  $x \in hds$ 
      then obtain  $ps$  where  $\text{hd } ps = x$   $\text{length } ps = 2$   $\text{adj-path } v \text{ } ps$   $\text{last } ps = v$ 
      using  $hds \text{ } l2\text{-eq-}v$  by auto
      thus  $\text{adjacent } v \text{ } x$ 
      by (metis (full-types) adj-path.simps(2) list.sel(1) length-0-conv
neq-Nil-conv
zero-neq-numeral)
    qed
  moreover have  $\bigwedge x. x \in \{n. \text{adjacent } v \text{ } n\} \implies x \in hds$ 
  proof –
    fix  $x$  assume  $x \in \{n. \text{adjacent } v \text{ } n\}$ 
    obtain  $ps$  where  $ps = [x, v]$  by auto
    hence  $\text{hd } ps = x$  and  $\text{length } ps = 2$  and  $\text{adj-path } v \text{ } ps$  and  $\text{last } ps = v$ 
    using  $\langle x \in \{n. \text{adjacent } v \text{ } n\} \rangle$  adjacent-sym by auto
    thus  $x \in hds$  by (metis (lifting, mono-tags) hds image-eqI l2-eq-v
mem-Collect-eq)
  qed
  ultimately show  $hds = \{n. \text{adjacent } v \text{ } n\}$  by auto
qed
moreover have inj-on  $\text{hd } l2\text{-eq-}v$  unfolding inj-on-def
proof (rule+)
  fix  $x \ y$  assume  $x \in l2\text{-eq-}v$   $y \in l2\text{-eq-}v$   $\text{hd } x = \text{hd } y$ 
  hence  $\text{length } x = 2$  and  $\text{last } x = \text{last } y$  and  $\text{length } y = 2$ 
  using  $l2\text{-eq-}v$  by auto
  hence  $x!1 = y!1$ 
  using last-conv-nth[of x] last-conv-nth[of y] by force
  moreover have  $x!0 = y!0$ 
  using  $\langle \text{hd } x = \text{hd } y \rangle$   $\langle \text{length } x = 2 \rangle$   $\langle \text{length } y = 2 \rangle$ 
  by (metis hd-conv-nth length-greater-0-conv)
  ultimately show  $x = y$  using  $\langle \text{length } x = 2 \rangle$   $\langle \text{length } y = 2 \rangle$ 
  using nth-equalityI[of x y]
  by (metis One-nat-def less-2-cases)
qed
  ultimately show  $\text{card } l2\text{-eq-}v = k$  using k-adj[OF v ∈ V] by (metis card-image)
qed
obtain  $l2\text{-neq-}v$  where  $l2\text{-neq-}v: l2\text{-neq-}v = \{ps. \text{length } ps = 2 \wedge \text{adj-path } v \text{ } ps \wedge \text{last } ps \neq v\}$  by auto
have  $\text{card } l2\text{-neq-}v = k * k - k$ 
proof –

```

obtain $l2-v$ **where** $l2-v:l2-v=\{ps. \text{length } ps=2 \wedge \text{adj-path } v \text{ } ps\}$ **by** *auto*
hence $\text{card } l2-v=k*k$ **using** $\text{path-count}[OF \text{ } k\text{-adj, of } v \text{ } 2] \langle 0 < k \rangle \langle \text{finite } V \rangle$
 $\langle v \in V \rangle$
by (*simp add: power2-eq-square*)
hence *finite* $l2-v$ **using** $\langle k > 0 \rangle$ **by** (*metis card.infinite mult-is-0 neq0-conv*)
moreover **have** $l2-v=l2\text{-neg-}v \cup l2\text{-eq-}v$ **using** $l2-v \text{ } l2\text{-neg-}v \text{ } l2\text{-eq-}v$ **by** *auto*
moreover **have** $l2\text{-neg-}v \cap l2\text{-eq-}v = \{\}$ **using** $l2\text{-neg-}v \text{ } l2\text{-eq-}v$ **by** *auto*
ultimately **have** $\text{card } l2\text{-neg-}v = \text{card } l2-v - \text{card } l2\text{-eq-}v$
by (*metis Int-commute Nat.add-0-right Un-commute card-Diff-subset-Int card-Un-Int*
card-gt-0-iff diff-add-inverse finite-Diff finite-Un inf-sup-absorb less-nat-zero-code)
thus $\text{card } l2\text{-neg-}v = k*k - k$ **using** $\langle \text{card } l2\text{-eq-}v = k \rangle$ **using** $\langle \text{card } l2-v = k*k \rangle$
by *auto*
qed
moreover **have** *bij-betw last* $l2\text{-neg-}v \{n. n \in V \wedge n \neq v\}$
proof –
have *last* ‘ $l2\text{-neg-}v = \{n. n \in V \wedge n \neq v\}$
proof –
have $\bigwedge x. x \in \text{last}' \text{ } l2\text{-neg-}v \implies x \in \{n. n \in V \wedge n \neq v\}$
proof
fix x **assume** $x \in \text{last}' \text{ } l2\text{-neg-}v$
then **obtain** ps **where** $\text{length } ps = 2 \text{ } \text{adj-path } v \text{ } ps \text{ } \text{last } ps = x \text{ } \text{last } ps \neq v$
using $l2\text{-neg-}v$ **by** *auto*
hence $(\text{last } ps) \in V$
by (*metis (full-types) adj-path-V last-in-set length-0-conv rev-subsetD zero-neq-numeral*)
thus $x \in V \wedge x \neq v$ **using** $\langle \text{last } ps = x \rangle \langle \text{last } ps \neq v \rangle$ **by** *auto*
qed
moreover **have** $\bigwedge x. x \in \{n. n \in V \wedge n \neq v\} \implies x \in \text{last}' \text{ } l2\text{-neg-}v$
proof –
fix x **assume** $x \in \{n \in V. n \neq v\}$
then **obtain** y **where** $\text{adjacent } v \text{ } y \text{ } \text{adjacent } x \text{ } y$
using *friend-assm[of v x]* $\langle v \in V \rangle$ **by** *auto*
hence $\text{adj-path } v \text{ } [y, x]$ **using** *adjacent-sym[of x y]* **by** *auto*
hence $[y, x] \in l2\text{-neg-}v$ **using** $l2\text{-neg-}v \text{ } x$ **by** *auto*
thus $x \in \text{last}' \text{ } l2\text{-neg-}v$ **by** (*metis imageI last.simps not-Cons-self2*)
qed
ultimately **show** *?thesis* **by** *fast*
qed
moreover **have** *inj-on last* $l2\text{-neg-}v$ **unfolding** *inj-on-def*
proof (*rule, rule, rule*)
fix $x \text{ } y$ **assume** $x \in l2\text{-neg-}v \text{ } y \in l2\text{-neg-}v \text{ } \text{last } x = \text{last } y$
hence $\text{length } x = 2$ **and** $\text{adj-path } v \text{ } x$ **and** $\text{last } x \neq v$ **and** $\text{length } y = 2$ **and**
 $\text{adj-path } v \text{ } y$
and $\text{last } y \neq v$
using $l2\text{-neg-}v$ **by** *auto*
obtain $x1 \text{ } x2 \text{ } y1 \text{ } y2$ **where** $x : x = [x1, x2]$ **and** $y : y = [y1, y2]$
proof –

```

{ fix l assume length l=2
  obtain h1 t where l=h1#t and length t=1
  using ⟨length l=2⟩ Suc-length-conv[of 1 l] by auto
  then obtain h2 where t=[h2]
  using Suc-length-conv[of 0 t] by auto
  have ∃ h1 h2. l=[h1,h2] using ⟨l=h1#t⟩ ⟨t=[h2]⟩ by auto }
thus ?thesis using that ⟨length x=2⟩ ⟨length y=2⟩ by metis
qed
hence x2≠v and y2≠v using ⟨last x≠v⟩ ⟨last y≠v⟩ by auto
moreover have adjacent v x1 and adjacent x2 x1 and x2∈V
  using ⟨adj-path v x⟩ x adjacent-sym by auto
moreover have adjacent v y1 and adjacent y2 y1 and y2∈V
  using ⟨adj-path v y⟩ y adjacent-sym by auto
ultimately have x1=y1 using friend-assm ⟨v∈V⟩
  by (metis ⟨last x = last y⟩ last-ConsL last-ConsR not-Cons-self2 x y)
thus x=y using x y ⟨last x = last y⟩ by auto
qed
ultimately show ?thesis unfolding bij-betw-def by auto
qed
hence card l2-neq-v = card {n. n∈V ∧ n≠v} by (metis bij-betw-same-card)
ultimately have card {n. n∈V ∧ n≠v}=k*k-k by auto
moreover have card V = card {n. n∈V ∧ n≠v} + card {v}
proof -
  have V={n. n∈V ∧ n≠v} ∪ {v} using ⟨v∈V⟩ by auto
  moreover have {n. n∈V ∧ n≠v} ∩ {v}={ } by auto
  ultimately show ?thesis
    using ⟨finite V⟩ card-Un-disjoint[of {n ∈ V. n ≠ v} {v}] finite-Un
    by auto
qed
ultimately show card V = k*k-k+1 by auto
qed

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

lemma rotate-diff:rotate m xs=rotate n xs ⟹rotate (m-n) xs = xs
proof (induct m arbitrary:n)
  case 0
  thus ?case by auto
next
  case (Suc m')

```

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

lemma (*in valid-unSimpGraph*) *exist-degree-two*:
 assumes *friend-assm*: $\bigwedge v u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. \text{adjacent } v \ n \ \wedge$
 adjacent } u \ n
 and *finite E and finite V and card V* ≥ 2
 shows $\exists v \in V. \text{degree } v \ G = 2$
proof (*rule ccontr*)
 assume $\neg (\exists v \in V. \text{degree } v \ G = 2)$
 hence $\bigwedge v. v \in V \implies \text{degree } v \ G \neq 2$ **by** *auto*
 obtain k **where** $k\text{-adj}$: $\bigwedge v. v \in V \implies \text{card } \{n. \text{adjacent } v \ n\} = k$ **using** *regular[OF friend-assm]*
 by (*metis* $\langle \neg (\exists v \in V. \text{degree } v \ G = 2) \rangle$ *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* $\langle \neg (\exists v \in V. \text{degree } v \ G = 2) \rangle$ *assms(2) de-*
 gree-two-windmill)
 have $k \neq 0$
 proof
 assume $k = 0$
 obtain $v3$ **where** *adjacent v1 v3* **using** *friend-assm[OF* $\langle v1 \in V \rangle \langle v2 \in V \rangle$
 $\langle v1 \neq v2 \rangle$ **by** *auto*
 hence $\text{card } \{n. \text{adjacent } v1 \ n\} \neq 0$ **using** *adjacent-finite[OF* $\langle \text{finite } E \rangle$
 by *auto*
 moreover have $\text{card } \{n. \text{adjacent } v1 \ n\} = 0$ **using** *k-adj[OF* $\langle v1 \in V \rangle$
 by (*metis* $\langle k = 0 \rangle$)
 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** $\langle \bigwedge v. v \in V \implies \text{degree } v \ G \neq 2 \rangle$ *degree-adjacent*
 $k\text{-adj}$
 by (*metis* $\langle v1 \in V \rangle$ *assms(2)*)

ultimately show *?thesis* by *auto*
 qed
 obtain T where $T:T=(\lambda l::nat. \{ps. length\ ps = l+1 \wedge adj\text{-}path\ (hd\ ps)\ (tl\ ps)\})$
 by *auto*
 have $T\text{-}count:\wedge l::nat. card\ (T\ l) = (k*k-k+1)*k^\wedge l$ using *card-partition'*
 proof –
 fix $l::nat$
 obtain ext where $ext:ext=(\lambda v\ ps. adj\text{-}path\ v\ (tl\ ps) \wedge hd\ ps=v \wedge length\ ps=l+1)$ by *auto*
 have $\forall v \in V. card\ \{ps. ext\ v\ ps\} = k^\wedge l$
 proof
 fix v assume $v \in V$
 have $\wedge ps. ps \in tl\ '\ \{ps. ext\ v\ ps\} \implies ps \in \{ps. length\ ps=l \wedge adj\text{-}path\ v\ ps\}$

 proof –
 fix ps assume $ps \in tl\ '\ \{ps. ext\ v\ ps\}$
 then obtain ps' where $adj\text{-}path\ v\ (tl\ ps')\ hd\ ps'=v\ length\ ps'=l+1$
 $ps=tl\ ps'$
 using *ext* by *auto*
 hence $adj\text{-}path\ v\ ps$ and $length\ ps=l$ by *auto*
 thus $ps \in \{ps. length\ ps=l \wedge adj\text{-}path\ v\ ps\}$ by *auto*
 qed
 moreover have $\wedge ps. ps \in \{ps. length\ ps=l \wedge adj\text{-}path\ v\ ps\} \implies ps \in tl\ '\ \{ps. ext\ v\ ps\}$
 proof –
 fix ps assume $ps \in \{ps. length\ ps = l \wedge adj\text{-}path\ v\ ps\}$
 hence $length\ ps=l$ and $adj\text{-}path\ v\ ps$ by *auto*
 moreover obtain ps' where $ps'=v\#\ ps$ by *auto*
 ultimately have $adj\text{-}path\ v\ (tl\ ps')$ and $hd\ ps'=v$ and $length\ ps'=l+1$
 thus $ps \in tl\ '\ \{ps. ext\ v\ ps\}$
 by *metis* $\langle ps' = v \#\ ps \rangle ext\ imageI\ mem\text{-}Collect\text{-}eq\ list.sel(3)$
 qed
 ultimately have $tl\ '\ \{ps. ext\ v\ ps\} = \{ps. length\ ps=l \wedge adj\text{-}path\ v\ ps\}$
 by *fast*
 moreover have $inj\text{-}on\ tl\ \{ps. ext\ v\ ps\}$ unfolding *inj-on-def*
 proof (*rule,rule,rule*)
 fix $x\ y$ assume $x \in Collect\ (ext\ v)\ y \in Collect\ (ext\ v)\ tl\ x = tl\ y$
 hence $hd\ x=hd\ y$ and $x \neq []$ and $y \neq []$ using *ext* by *auto*
 thus $x=y$ using $\langle tl\ x = tl\ y \rangle$ by (*metis list.sel(1,3) list.exhaust*)
 qed
 moreover have $card\ \{ps. length\ ps=l \wedge adj\text{-}path\ v\ ps\} = k^\wedge l$
 using *path-count[OF k-adj,of v l] <4 ≤ k> <v ∈ V> assms(3)*
 by *auto*
 ultimately show $card\ \{ps. ext\ v\ ps\} = k^\wedge l$ by (*metis card-image*)
 qed
 moreover have $\forall v1\ v2. v1 \neq v2 \longrightarrow \{n. ext\ v1\ n\} \cap \{n. ext\ v2\ n\} = \{\}$
 using *ext* by *auto*
 moreover have $(\bigcup v \in V. \{n. ext\ v\ n\}) = T\ l$

proof –
have $\bigwedge ps. ps \in (\bigcup v \in V. \{n. ext\ v\ n\}) \implies ps \in T\ l$ **using** T
proof –
fix ps **assume** $ps \in (\bigcup v \in V. \{n. ext\ v\ n\})$
then obtain v **where** $v \in V$ $adj\text{-}path\ v\ (tl\ ps)$ $hd\ ps = v$ $length\ ps = l$
+ 1
using ext **by** $auto$
hence $length\ ps = l + 1$ **and** $adj\text{-}path\ (hd\ ps)\ (tl\ ps)$ **by** $auto$
thus $ps \in T\ l$ **using** T **by** $auto$
qed
moreover have $\bigwedge ps. ps \in T\ l \implies ps \in (\bigcup v \in V. \{n. ext\ v\ n\})$
proof –
fix ps **assume** $ps \in T\ l$
hence $length\ ps = l + 1$ **and** $adj\text{-}path\ (hd\ ps)\ (tl\ ps)$ **using** T **by** $auto$
moreover then obtain v **where** $v = hd\ ps$ $v \in V$
by $(metis\ adj\text{-}path.simps(1)\ adj\text{-}path.simps(2)\ adjacent\text{-}V(1))$
list.exhaust
ultimately show $ps \in (\bigcup v \in V. \{n. ext\ v\ n\})$ **using** ext **by** $auto$
qed
ultimately show $?thesis$ **by** $auto$
qed
ultimately have $card\ (T\ l) = card\ V * k^\wedge l$
using $card\text{-}partition'$ $[of\ V\ ext\ k^\wedge l]$ $\langle 4 \leq k \rangle$ $assms(3)$ $mult.commute$
nat-one-le-power
by $auto$
moreover have $card\ V = (k * k - k + 1)$
using $total\text{-}v\text{-}num$ $[OF\ friend\text{-}assm, of\ k]$ $k\text{-}adj\ degree\text{-}adjacent$ $\langle finite\ E \rangle$
 $\langle finite\ V \rangle$
 $\langle card\ V \geq 2 \rangle$ $\langle 4 \leq k \rangle$ $card\text{-}gt\text{-}0\text{-}iff$
by $force$
ultimately show $card\ (T\ l) = (k * k - k + 1) * k^\wedge l$ **by** $auto$
qed
obtain C **where** $C : C = (\lambda l :: nat. \{ps. length\ ps = l + 1 \wedge adj\text{-}path\ (hd\ ps)\ (tl\ ps) \wedge adjacent\ (last\ ps)\ (hd\ ps)\})$ **by** $auto$
obtain $C\text{-}star$ **where** $C\text{-}star : C\text{-}star = (\lambda l :: nat. \{ps. length\ ps = l + 1 \wedge adj\text{-}path\ (hd\ ps)\ (tl\ ps) \wedge (last\ ps) = (hd\ ps)\})$ **by** $auto$
have $\bigwedge l :: nat. card\ (C\ (l + 1)) = k * card\ (C\text{-}star\ l) + card\ (T\ l - C\text{-}star\ l)$
proof –
fix $l :: nat$
have $C\ (l + 1) = \{ps. length\ ps = l + 2 \wedge adj\text{-}path\ (hd\ ps)\ (tl\ ps) \wedge adjacent\ (last\ ps)\ (hd\ ps) \wedge last\ (butlast\ ps) = hd\ ps\} \cup \{ps. length\ ps = l + 2 \wedge adj\text{-}path\ (hd\ ps)\ (tl\ ps) \wedge adjacent\ (last\ ps)\ (hd\ ps) \wedge last\ (butlast\ ps) \neq hd\ ps\}$ **using** C **by** $auto$
moreover have $\{ps. length\ ps = l + 2 \wedge adj\text{-}path\ (hd\ ps)\ (tl\ ps) \wedge adjacent\ (last\ ps)\ (hd\ ps) \wedge last\ (butlast\ ps) = hd\ ps\} \cap \{ps. length\ ps = l + 2 \wedge adj\text{-}path\ (hd\ ps)\ (tl\ ps) \wedge adjacent\ (last\ ps)\ (hd\ ps) \wedge last\ (butlast\ ps) \neq hd\ ps\}$

$adjacent (last\ ps) (hd\ ps) \wedge last (butlast\ ps) \neq hd\ ps \} = \{ \}$ **by auto**
moreover have $finite (C\ (l+1))$
proof –
have $C\ (l+1) \subseteq T\ (l+1)$ **using** $C\ T$ **by auto**
moreover have $(k * k - k + 1) * k \wedge (l + 1) \neq 0$ **using** $\langle k \geq 4 \rangle$ **by auto**
hence $finite (T\ (l+1))$ **using** $T\text{-count}[of\ l+1]$ **by** $(metis\ card.infinite)$
ultimately show $?thesis$ **by** $(metis\ finite-subset)$
qed
ultimately have $card (C\ (l+1)) = card \{ps.\ length\ ps = l+2 \wedge adj\text{-}path (hd\ ps) (tl\ ps) \wedge adjacent (last\ ps) (hd\ ps) \wedge last (butlast\ ps) = hd\ ps \} + card \{ps.\ length\ ps = l+2 \wedge adj\text{-}path (hd\ ps) (tl\ ps) \wedge adjacent (last\ ps) (hd\ ps) \wedge last (butlast\ ps) \neq hd\ ps \}$
using $card\text{-}Un\text{-}disjoint[of\ \{ps.\ length\ ps = l + 2 \wedge adj\text{-}path (hd\ ps) (tl\ ps) \wedge adjacent (last\ ps) (hd\ ps) \wedge last (butlast\ ps) = hd\ ps \} \{ps.\ length\ ps = l + 2 \wedge adj\text{-}path (hd\ ps) (tl\ ps) \wedge adjacent (last\ ps) (hd\ ps) \wedge last (butlast\ ps) \neq hd\ ps \}]$ $finite\text{-}Un$ **by auto**
moreover have $card \{ps.\ length\ ps = l+2 \wedge adj\text{-}path (hd\ ps) (tl\ ps) \wedge adjacent (last\ ps) (hd\ ps) \wedge last (butlast\ ps) = hd\ ps \} = k * card (C\text{-}star\ l)$
proof –
obtain ext **where** $ext: ext = (\lambda ps\ ps'. ps' \neq [] \wedge (butlast\ ps' = ps) \wedge adj\text{-}path (hd\ ps') (tl\ ps'))$ **by auto**
have $\forall ps \in (C\text{-}star\ l). card \{ps'. ext\ ps\ ps'\} = k$
proof
fix ps **assume** $ps \in C\text{-}star\ l$
hence $length\ ps = l + 1$ **and** $adj\text{-}path (hd\ ps) (tl\ ps)$ **and** $last\ ps = hd\ ps$
using $C\text{-}star$ **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 $\bigwedge x. x \in app\ 'qs \implies x \in \{ps'. ext\ ps\ ps'\}$
proof
fix x **assume** $x \in app\ 'qs$
then obtain y **where** $adjacent (last\ ps) y$ $x = ps@[y]$ **using** qs
moreover hence $adj\text{-}path (hd\ x) (tl\ x)$
by $(cases\ tl\ ps = [],\ metis\ adj\text{-}path.\ simps(1)\ adj\text{-}path.\ simps(2)\ adjacent\text{-}V(2)\ append\text{-}Nil\ list.\ sel(1,3)\ hd\text{-}append\ snoc\text{-}eq\text{-}iff\text{-}butlast)$
 $tl\text{-}append2,\ metis\ \langle adj\text{-}path (hd\ ps) (tl\ ps) \rangle\ adj\text{-}path\text{-}app$
 $last\text{-}tl\ list.\ sel(2)\ tl\text{-}append2)$
ultimately show $ext\ ps\ x$ **using** ext **by** $(metis\ snoc\text{-}eq\text{-}iff\text{-}butlast)$
qed

```

moreover have  $\bigwedge x. x \in \{ps'. ext\ ps\ ps'\} \implies x \in app\ 'qs$ 
proof –
  fix  $x$  assume  $x \in \{ps'. ext\ ps\ ps'\}$ 
  hence  $x \neq []$  and  $butlast\ x = ps$  and  $adj\text{-}path\ (hd\ x)\ (tl\ x)$ 
  using  $ext$  by  $auto$ 
  have  $adjacent\ (last\ ps)\ (last\ x)$ 
  proof ( $cases\ length\ ps = 1$ )
    case  $True$ 
      hence  $length\ x = 2$  using  $\langle butlast\ x = ps \rangle$  by  $auto$ 
      then obtain  $x1\ t1$  where  $x = x1 \# t1$  and  $length\ t1 = 1$ 
      using  $Suc\text{-}length\text{-}conv[of\ 1\ x]$  by  $auto$ 
      then obtain  $x2$  where  $t1 = [x2]$ 
      using  $Suc\text{-}length\text{-}conv[of\ 0\ t1]$  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  $\langle adj\text{-}path\ (hd\ x)\ (tl\ x) \rangle$   $\langle butlast\ x = ps \rangle$  by  $auto$ 
    next
      case  $False$ 
      hence  $tl\ ps \neq []$ 
      by ( $metis\ \langle length\ ps = l + 1 \rangle\ add\text{-}0\text{-}iff\ add\text{-}diff\text{-}cancel\text{-}left'$ 
         $length\text{-}0\text{-}conv\ length\text{-}tl\ add.\ commute$ )
      moreover have  $adj\text{-}path\ (hd\ x)\ (tl\ ps\ @\ [last\ x])$ 
      using  $\langle adj\text{-}path\ (hd\ x)\ (tl\ x) \rangle$   $\langle butlast\ x = ps \rangle$   $\langle x \neq [] \rangle$ 
      by ( $metis\ append\text{-}butlast\text{-}last\text{-}id\ calculation\ list.sel(2)$ )
      ultimately have  $adjacent\ (last\ (tl\ ps))\ (last\ x)$ 
      using  $adj\text{-}path\text{-}app'[of\ hd\ x\ tl\ ps\ last\ x]$ 
      by  $auto$ 
      thus  $adjacent\ (last\ ps)\ (last\ x)$  by ( $metis\ \langle tl\ ps \neq [] \rangle\ last\text{-}tl$ )
      qed
    thus  $x \in app\ 'qs$  using  $app\ qs$ 
    by ( $metis\ \langle butlast\ x = ps \rangle\ \langle x \neq [] \rangle\ append\text{-}butlast\text{-}last\text{-}id$ )
   $mem\text{-}Collect\text{-}eq$ 
   $rev\text{-}image\text{-}eqI$ 
  qed
  ultimately show  $?thesis$  by  $auto$ 
  qed
  moreover have  $inj\text{-}on\ app\ qs$  using  $app\ unfolding\ inj\text{-}on\text{-}def$  by
 $auto$ 
  moreover have  $last\ ps \in V$ 
  using  $\langle length\ ps = l + 1 \rangle$   $\langle adj\text{-}path\ (hd\ ps)\ (tl\ ps) \rangle$   $adj\text{-}path\text{-}V$ 
  by ( $metis\ \langle last\ ps = hd\ ps \rangle\ adj\text{-}path.\ simps(1)\ last\text{-}in\text{-}set\ last\text{-}tl$ 
 $subset\text{-}code(1)$ )
  hence  $card\ qs = k$  using  $qs\ k\text{-}adj$  by  $auto$ 
  ultimately show  $card\ \{ps'. ext\ ps\ ps'\} = k$  by ( $metis\ card\text{-}image$ )
  qed
  moreover have  $finite\ (C\text{-}star\ l)$ 
  proof –
  have  $C\text{-}star\ l \subseteq T\ l$  using  $C\text{-}star\ T$  by  $auto$ 

```


moreover have $(k * k - k + 1) * k \wedge l \neq 0$ **using** $\langle k \geq 4 \rangle$ **by** *auto*
hence *finite* $(T\ l)$ **using** *T-count[of l]* **by** $(metis\ card.infinite)$
ultimately show *?thesis* **by** $(metis\ finite-subset)$
qed
moreover have $\forall ps1\ ps2. ps1 \neq ps2 \longrightarrow \{ps'.\ ext\ ps1\ ps'\} \cap \{ps'.\ ext\ ps2\ ps'\} = \{\}$
using *ext* **by** *auto*
moreover have $(\bigcup ps \in (C\text{-star}\ l). \{ps'.\ ext\ ps\ ps'\}) = \{ps.\ length\ ps = l+2$
 $\wedge\ adj\text{-path}\ (hd\ ps)\ (tl\ ps) \wedge\ adjacent\ (last\ ps)\ (hd\ ps) \wedge\ last\ (butlast\ ps)=hd\ ps\}$
proof –
have $\bigwedge x. x \in (\bigcup ps \in (C\text{-star}\ l). \{ps'.\ ext\ ps\ ps'\}) \implies x \in \{ps.\ length\ ps = l+2$
 $\wedge\ adj\text{-path}\ (hd\ ps)\ (tl\ ps) \wedge\ adjacent\ (last\ ps)\ (hd\ ps) \wedge\ last\ (butlast\ ps)=hd\ ps\}$
proof
fix x **assume** $x \in (\bigcup ps \in C\text{-star}\ l. \{ps'.\ ext\ ps\ ps'\})$
then obtain ps **where** $ps \in C\text{-star}\ l$ **ext** $ps\ x$ **by** *auto*
hence $length\ ps = l + 1$ **and** $adj\text{-path}\ (hd\ ps)\ (tl\ ps)$ **and** $last\ ps = hd\ ps$
and $x \neq []$ **and** $butlast\ x = ps$ $adj\text{-path}\ (hd\ x)\ (tl\ x)$
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 $adj\text{-path}\ (hd\ x)\ (tl\ x)$ **by** $(metis\ \langle adj\text{-path}\ (hd\ x)\ (tl\ x) \rangle)$
moreover have $adjacent\ (last\ x)\ (hd\ x)$
proof –
have $length\ x \geq 2$ **using** $\langle length\ x = l + 2 \rangle$ **by** *auto*
hence $adjacent\ (last\ (butlast\ x))\ (last\ x)$ **using** $\langle adj\text{-path}\ (hd\ x)\ (tl\ x) \rangle$
by $(induct\ x, auto, metis\ adj\text{-path.simps}(2)\ append\text{-butlast}\text{-last}\text{-id}\ append\text{-eq}\text{-Cons}\text{-conv}, metis\ adj\text{-path}\text{-app}'\ append\text{-butlast}\text{-last}\text{-id})$
hence $adjacent\ (last\ ps)\ (last\ x)$ **using** $\langle butlast\ x = ps \rangle$ **by** *auto*
hence $adjacent\ (hd\ ps)\ (last\ x)$ **using** $\langle last\ ps = hd\ ps \rangle$ **by** *auto*
hence $adjacent\ (hd\ x)\ (last\ x)$
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 $(metis\ \langle butlast\ x = ps \rangle \langle last\ ps = hd\ ps \rangle \langle x \neq [] \rangle\ adj\text{-no}\text{-loop}\ butlast.simps(2)\ calculation(3)\ list.sel(1)\ last\text{-ConsL}\ neq\text{-Nil}\text{-conv})$
ultimately show $length\ x = l + 2 \wedge\ adj\text{-path}\ (hd\ x)\ (tl\ x)$
 $\wedge\ adjacent\ (last\ x)\ (hd\ x) \wedge\ last\ (butlast\ x) = hd\ x$

by *auto*
 qed
moreover have $\bigwedge x. x \in \{ps. \text{length } ps = l+2 \wedge \text{adj-path } (hd \ ps) \ (tl \ ps) \wedge \text{adjacent } (last \ ps) \ (hd \ ps) \wedge last \ (butlast \ ps) = hd \ ps\} \implies$
 $x \in (\bigcup ps \in (C\text{-star } l). \{ps'. \text{ext } ps \ ps'\})$
proof –
fix x **assume** $x \in \{ps. \text{length } ps = l+2 \wedge \text{adj-path } (hd \ ps) \ (tl \ ps) \wedge \text{adjacent } (last \ ps) \ (hd \ ps) \wedge last \ (butlast \ ps) = hd \ ps\}$
hence $\text{length } x = l+2$ **and** $\text{adj-path } (hd \ x) \ (tl \ x)$ **and** $\text{adjacent } (last$
 $x) \ (hd \ x)$
and $last \ (butlast \ x) = hd \ x$ **by** *auto*
obtain ps **where** $ps:ps = butlast \ x$ **by** *auto*
have $ps \in C\text{-star } l$
proof –
have $\text{length } ps = l + 1$ **using** $ps \langle \text{length } x = l+2 \rangle$ **by** *auto*
moreover have $hd \ ps = hd \ x$
using $ps \langle \text{length } x = l+2 \rangle$
by $(metis \ (full\text{-types}) \ \langle \text{adjacent } (last \ x) \ (hd \ x) \ \rangle \ \text{adjacent-no-loop}$
 $\text{append-Nil } \text{append-butlast-last-id } butlast.simps(1) \ list.sel(1)$
 $hd\text{-append2})$
hence $\text{adj-path } (hd \ ps) \ (tl \ ps)$ **using** *adj-path-butlast*
by $(metis \ \langle \text{adj-path } (hd \ x) \ (tl \ x) \ \rangle \ butlast-tl \ ps)$
moreover have $last \ ps = hd \ ps$
by $(metis \ \langle hd \ ps = hd \ x \ \rangle \ \langle last \ (butlast \ x) = hd \ x \ \rangle \ ps)$
ultimately show *?thesis* **using** *C-star* **by** *auto*
 qed
moreover have $\text{ext } ps \ x$ **using** *ext*
by $(metis \ \langle \text{adj-path } (hd \ x) \ (tl \ x) \ \rangle \ \langle \text{adjacent } (last \ x) \ (hd \ x) \ \rangle$
 $\langle last \ (butlast \ x) = hd \ x \ \rangle \ \text{adjacent-no-loop } butlast.simps(1) \ ps)$
ultimately show $x \in (\bigcup ps \in (C\text{-star } l). \{ps'. \text{ext } ps \ ps'\})$ **by** *auto*
 qed
ultimately show *?thesis* **by** *fast*
 qed
ultimately show *?thesis* **using** *card-partition'* $[of \ C\text{-star } l \ \text{ext } k] \ \langle k \geq 4 \rangle$
by *auto*
 qed
moreover have $\text{card } \{ps. \text{length } ps = l+2 \wedge \text{adj-path } (hd \ ps) \ (tl \ ps) \wedge$
 $\text{adjacent } (last \ ps) \ (hd \ ps) \wedge last \ (butlast \ ps) \neq hd \ ps\} = \text{card } (T \ l - C\text{-star}$
 $l)$
proof –
obtain app **where** $app:app = (\lambda ps. ps @ [SOME \ n. \ \text{adjacent } (last \ ps) \ n \wedge$
 $\text{adjacent } (hd \ ps) \ n])$
by *auto*
have $\bigwedge x. x \in app \ (T \ l - C\text{-star } l) \implies x \in \{ps. \text{length } ps = l+2 \wedge \text{adj-path}$
 $(hd \ ps) \ (tl \ ps) \wedge$
 $\text{adjacent } (last \ ps) \ (hd \ ps) \wedge last \ (butlast \ ps) \neq hd \ ps\}$
proof
fix x **assume** $x \in app \ (T \ l - C\text{-star } l)$

then obtain ps where $length\ ps = l + 1$ $adj\text{-}path\ (hd\ ps)\ (tl\ ps)$ $last\ ps \neq hd\ ps$

$x = app\ ps$
using $T\ C\text{-}star$ **by** $auto$
hence $last\ ps \in V$
using $adj\text{-}path\text{-}V[OF\ \langle adj\text{-}path\ (hd\ ps)\ (tl\ ps) \rangle]$
by $(cases\ ps)\ auto$
hence $\exists n.\ adjacent\ (last\ ps)\ n \wedge adjacent\ (hd\ ps)\ n$
using $adj\text{-}path\text{-}V[OF\ \langle adj\text{-}path\ (hd\ ps)\ (tl\ ps) \rangle]\ \langle last\ ps \neq hd\ ps \rangle$
 $friend\text{-}assm[of\ last\ ps\ hd\ ps]$
by $auto$
moreover have $last\ x = (SOME\ n.\ adjacent\ (last\ ps)\ n \wedge adjacent\ (hd\ ps)\ n)$

using $app\ \langle x = app\ ps \rangle$ **by** $auto$
ultimately have $adjacent\ (last\ ps)\ (last\ x)$ **and** $adjacent\ (hd\ ps)\ (last\ x)$

using $someI\text{-}ex$ **by** $(metis\ (lifting))+$
have $hd\ x = hd\ ps$ **using** $\langle x = app\ ps \rangle\ \langle length\ ps = l + 1 \rangle\ app$
by $(cases\ ps)\ auto$
have $length\ x = l + 2$ **using** $\langle x = app\ ps \rangle\ \langle length\ ps = l + 1 \rangle\ app$ **by** $auto$
moreover have $adj\text{-}path\ (hd\ x)\ (tl\ x)$
proof –
have $last\ (tl\ ps) = last\ ps$ **using** $\langle length\ ps = l + 1 \rangle$
by $(metis\ \langle last\ ps \neq hd\ ps \rangle\ list.sel(1,3)\ last\text{-}ConsL\ last\text{-}tl\ neq\text{-}Nil\text{-}conv)$

moreover have $length\ ps \neq 1$ **using** $\langle last\ ps \neq hd\ ps \rangle$
by $(metis\ Suc\text{-}eq\text{-}plus1\text{-}left\ gen\text{-}length\text{-}code(1)\ gen\text{-}length\text{-}def\ list.sel(1)\ last\text{-}ConsL\ length\text{-}Suc\text{-}conv\ neq\text{-}Nil\text{-}conv)$

hence $tl\ ps \neq []$ **using** $\langle length\ ps = l + 1 \rangle$
by $(auto\ simp:\ length\text{-}Suc\text{-}conv)$
ultimately have $adj\text{-}path\ (hd\ ps)\ (tl\ ps\ @\ [last\ x])$
using $adj\text{-}path\text{-}app[OF\ \langle adj\text{-}path\ (hd\ ps)\ (tl\ ps) \rangle,\ of\ last\ x]\ \langle adjacent\ (last\ ps)\ (last\ x) \rangle$
by $auto$
moreover have $tl\ ps\ @\ [last\ x] = tl\ x$
using $\langle x = app\ ps \rangle\ app$
by $(metis\ \langle last\ x = (SOME\ n.\ adjacent\ (last\ ps)\ n \wedge adjacent\ (hd\ ps)\ n) \rangle\ \langle tl\ ps \neq [] \rangle\ list.sel(2)\ tl\text{-}append2)$

ultimately show $?thesis$ **using** $\langle hd\ x = hd\ ps \rangle$ **by** $auto$
qed
moreover have $adjacent\ (last\ x)\ (hd\ x)$
using $\langle hd\ x = hd\ ps \rangle\ \langle adjacent\ (hd\ ps)\ (last\ x) \rangle\ adjacent\text{-}sym$ **by** $auto$
moreover have $last\ (butlast\ x) \neq hd\ x$
using $\langle last\ ps \neq hd\ ps \rangle\ \langle hd\ x = hd\ ps \rangle$
by $(metis\ \langle x = app\ ps \rangle\ app\ butlast\text{-}snoc)$
ultimately show $length\ x = l + 2 \wedge adj\text{-}path\ (hd\ x)\ (tl\ x) \wedge adjacent\ (last\ x)\ (hd\ x)$

$\wedge \text{last } (\text{butlast } x) \neq \text{hd } x$
by auto
qed
moreover have $\wedge x. x \in \{ps. \text{length } ps = l+2 \wedge \text{adj-path } (\text{hd } ps) (\text{tl } ps) \wedge$
 $\text{adjacent } (\text{last } ps) (\text{hd } ps) \wedge \text{last } (\text{butlast } ps) \neq \text{hd } ps\} \implies x \in \text{app } (T \ l \ -$
C-star l)
proof -
fix x assume $x \in \{ps. \text{length } ps = l+2 \wedge \text{adj-path } (\text{hd } ps) (\text{tl } ps) \wedge$
 $\text{adjacent } (\text{last } ps) (\text{hd } ps) \wedge \text{last } (\text{butlast } ps) \neq \text{hd } ps\}$
hence $\text{length } x = l+2$ **and** $\text{adj-path } (\text{hd } x) (\text{tl } x)$ **and** $\text{adjacent } (\text{last } x)$
(hd x)
and $\text{last } (\text{butlast } x) \neq \text{hd } x$
by auto
hence $\text{butlast } x \in T \ l \ - \ \text{C-star } l$
proof -
have $\text{length } (\text{butlast } x) = l + 1$
using $\langle \text{length } x = l + 2 \rangle \text{length-butlast}$ **by auto**
moreover have $\text{hd } (\text{butlast } x) = \text{hd } x$
using $\langle \text{length } x = l+2 \rangle$
by *(metis append-butlast-last-id butlast.simps(1) calculation*
diff-add-inverse
diff-cancel2 hd-append length-butlast add commute num.distinct(1)
one-eq-numeral-iff)
hence $\text{adj-path } (\text{hd } (\text{butlast } x)) (\text{tl } (\text{butlast } x))$
using $\langle \text{adj-path } (\text{hd } x) (\text{tl } x) \rangle$ **by** *(metis adj-path-butlast butlast-tl)*
moreover have $\text{last } (\text{butlast } x) \neq \text{hd } (\text{butlast } x)$
using $\langle \text{last } (\text{butlast } x) \neq \text{hd } x \rangle \langle \text{hd } (\text{butlast } x) = \text{hd } x \rangle$ **by auto**
ultimately show *?thesis using T C-star by auto*
qed
moreover have $\text{app } (\text{butlast } x) = x$ **using** *app*
proof -
have $\text{last } (\text{butlast } x) \in V$
proof *(cases length x ≥ 3)*
case True
hence $\text{last } (\text{butlast } x) \in \text{set } (\text{tl } x)$
proof *(induct x)*
case Nil
thus *?case by auto*
next
case *(Cons x1 t1)*
have $\text{length } t1 < 3 \implies ?case$
proof -
assume $\text{length } t1 < 3$
hence $\text{length } t1 = 2$ **using** $\langle 3 \leq \text{length } (x1 \# t1) \rangle$ **by auto**
then obtain $x2 \ t2$ **where** $t1 = x2 \# t2$ $\text{length } t2 = 1$
using *Suc-length-conv[of 1 t1]* **by auto**
then obtain $x3$ **where** $t2 = [x3]$
using *Suc-length-conv[of 0 t2]* **by auto**

```

      have  $t1=[x2,x3]$  using  $\langle t1=x2\#t2 \rangle \langle t2=[x3] \rangle$  by auto
      thus ?case by auto
    qed
  moreover have  $length\ t1 \geq 3 \implies ?case$ 
  proof -
    assume  $length\ t1 \geq 3$ 
    hence  $last\ (butlast\ t1) \in set\ (tl\ t1)$ 
      using Cons.hyps by auto
    thus ?case
      by (metis butlast.simps(2) in-set-butlastD last.simps
          length-butlast length-greater-0-conv length-pos-if-in-set
          length-tl list.sel(3))
    qed
  ultimately show ?case by force
  qed
  thus ?thesis using adj-path-V[OF  $\langle adj-path\ (hd\ x)\ (tl\ x) \rangle$ ] by
auto

next
case False
hence  $length\ x=2$  using  $\langle length\ x=l+2 \rangle$  by auto
then obtain  $x1\ x2$  where  $x=[x1,x2]$ 
  proof -
    obtain  $x1\ t1$  where  $x=x1\ \#t1$   $length\ t1=1$ 
      using Suc-length-conv[of  $1\ x$ ]  $\langle length\ x=2 \rangle$  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 ?thesis using that by auto
  qed
  hence  $last\ (butlast\ x)=hd\ x$  by auto
  thus ?thesis using adj-path-V'[OF  $\langle adj-path\ (hd\ x)\ (tl\ x) \rangle$ ] by
auto

  qed
  moreover have  $hd\ (butlast\ x)=hd\ x$  using  $\langle length\ x=l+2 \rangle$ 
    by (metis  $\langle adjacent\ (last\ x)\ (hd\ x) \rangle$  adjacent-no-loop ap-
pend-butlast-last-id
        butlast.simps(1) list.sel(1) hd-append)
  hence  $hd\ (butlast\ x) \in V$  using adj-path-V'[OF  $\langle adj-path\ (hd\ x)\ (tl$ 
 $x) \rangle$ ] by auto
  moreover have  $last\ (butlast\ x) \neq hd\ (butlast\ x)$ 
    using  $\langle last\ (butlast\ x) \neq hd\ x \rangle \langle hd\ (butlast\ x) = hd\ x \rangle$  by auto
  ultimately have  $\exists! n. adjacent\ (last\ (butlast\ x))\ n \wedge adjacent\ (hd$ 
 $(butlast\ x))\ n$ 
    using friend-assm by auto
  moreover have  $length\ x \geq 2$  using  $\langle length\ x=l+2 \rangle$  by auto
  hence  $adjacent\ (last\ (butlast\ x))\ (last\ x)$ 
    using  $\langle adj-path\ (hd\ x)\ (tl\ x) \rangle$ 
  by (induct  $x, auto, metis$  (full-types) adj-path.simps(2) append-Nil

```

```

    append-butlast-last-id, metis adj-path-app' append-butlast-last-id)
moreover have adjacent (hd (butlast x)) (last x)
using ⟨adjacent (last x) (hd x)⟩ ⟨hd (butlast x)=hd x⟩ adjacent-sym
by auto
ultimately have (SOME n. adjacent (last (butlast x)) n
  ∧ adjacent (hd (butlast x)) n) = last x
using some1-equality by fast
moreover have x=(butlast x)@[last x]
by (metis ⟨adjacent (last (butlast x)) (last x)⟩ adjacent-no-loop
  append-butlast-last-id butlast.simps(1))
ultimately show ?thesis using app by auto
qed
ultimately show x∈app“(T l - C-star l) by (metis image-iff)
qed
ultimately have app“(T l - C-star l)={ps. length ps = l+2 ∧ adj-path
(hd ps) (tl ps) ∧
  adjacent (last ps) (hd ps) ∧ last (butlast ps)≠hd ps} by fast
moreover have inj-on app (T l - C-star l) using app unfolding inj-on-def
by auto
ultimately show ?thesis by (metis card-image)
qed
ultimately show card (C (l + 1)) = k * card (C-star l) + card (T l -
C-star l) by auto
qed
hence ∧l::nat. card (C (l+1)) mod (k-(1::nat))=1
proof -
  fix l::nat
  have C-star l ⊆ T l using C-star T by auto
  moreover have card (T l)≠0 using T-count ⟨k≥4⟩ by auto
  hence finite (T l) using ⟨k≥4⟩ by (metis card.infinite)
  ultimately have card (T l - C-star l)=card(T l) - card(C-star l)
  by (metis card-Diff-subset rev-finite-subset)
  hence card (C (l + 1))=k*card (C-star l) + (card (T l) - card (C-star l))
  using ⟨∧l::nat. card (C (l+1)) = k* card (C-star l) + card (T l - C-star
l)⟩
  by auto
also have ...=k*card (C-star l) + card (T l) - card (C-star l)
proof -
  have card (T l) ≥ card (C-star l)
  using ⟨C-star l ⊆ T l⟩ ⟨finite (T l)⟩ by (metis card-mono)
  thus ?thesis by auto
qed
also have ...=k*card (C-star l) - card (C-star l) + card (T l)
proof -
  have card (T l) ≥ card (C-star l)
  using ⟨C-star l ⊆ T l⟩ ⟨finite (T l)⟩ by (metis card-mono)
  moreover have k*card (C-star l) ≥ card (C-star l) using ⟨k≥4⟩ by auto
  ultimately show ?thesis by auto
qed

```

also have ...= $(k-(1::nat))*card(C\text{-star } l)+card(T\ l)$ **using** $\langle k \geq 4 \rangle$
by (*metis monoid-mult-class.mult.left-neutral diff-mult-distrib*)
finally have $card(C(l+1))=(k-(1::nat))*card(C\text{-star } l)+card(T\ l)$.
hence $card(C(l+1))\ mod\ (k-(1::nat)) = card(T\ l)\ mod\ (k-(1::nat))$ **using**
 $\langle k \geq 4 \rangle$
by (*metis mod-mult-self3 mult.commute*)
also have ...= $((k*k-k+1)*k^l)\ mod\ (k-(1::nat))$ **using** *T-count* **by** *auto*
also have ...= $((k-(1::nat))*k+1)*k^l\ mod\ (k-(1::nat))$
proof -
have $k*k-k+1=(k-(1::nat))*k+1$ **using** $\langle k \geq 4 \rangle$ **by** (*metis diff-mult-distrib*
nat-mult-1)
thus *?thesis* **by** *auto*
qed
also have ...= $1*k^l\ mod\ (k-(1::nat))$
by (*metis mod-mult-right-eq mod-mult-self1 add.commute mult.commute*)
also have ...= $k^l\ mod\ (k-(1::nat))$ **by** *auto*
also have ...= $(k-(1::nat)+1)^l\ mod\ (k-(1::nat))$ **using** $\langle k \geq 4 \rangle$ **by** *auto*
also have ...= $1^l\ mod\ (k-(1::nat))$ **by** (*metis mod-add-self2 add.commute*
power-mod)
also have ...= $1\ mod\ (k-(1::nat))$ **by** *auto*
also have ...= 1 **using** $\langle k \geq 4 \rangle$ **by** *auto*
finally show $card(C(l+1))\ mod\ (k-(1::nat)) = 1$.
qed
obtain $p::nat$ **where** *prime p p dvd (k-(1::nat))* **using** $\langle k \geq 4 \rangle$
by (*metis Suc-eq-plus1 Suc-numeral add-One-commute eq-iff le-diff-conv numeral-le-iff*
one-le-numeral one-plus-BitM prime-factor-nat semiring-norm(69) semiring-norm(71))
hence $p\text{-minus-1}:p-(1::nat)+1=p$
by (*metis add-diff-inverse add.commute not-less-iff-gr-or-eq prime-nat-iff*)
hence $*$: $\bigwedge l::nat. card(C(l+1))\ mod\ p=1$
using $\langle \bigwedge l::nat. card(C(l+1))\ mod\ (k-(1::nat))=1 \rangle$ *mod-mod-cancel[OF $\langle p$*
dvd (k-(1::nat))\rangle
 \langle *prime p* \rangle
by (*metis mod-if prime-gt-1-nat*)
have $card(C(p-1))\ mod\ p = 1$
proof (*cases 2 ≤ p*)
case *True* **with** $*$ [*of p - 2*] **show** *?thesis*
by (*metis Nat.add-diff-assoc2 add-le-cancel-right diff-diff-left one-add-one*
p-minus-1)
next
case *False* **with** $*$ [*of p - 2*] \langle *prime p* \rangle *prime-ge-2-nat* **show** *?thesis*
by *blast*
qed
moreover have $card(C(p-(1::nat)))\ mod\ p=0$ **using** *C*
proof -
have *closure1*: $\bigwedge x. x \in C(p-(1::nat)) \implies rotate1\ x \in C(p-(1::nat))$
proof -
fix x **assume** $x \in C(p-(1::nat))$

hence $\text{length } x = p$ **and** $\text{adj-path } (\text{hd } x) (\text{tl } x)$ **and** $\text{adjacent } (\text{last } x) (\text{hd } x)$
x)
using $C \text{ p-minus-1}$ **by** *auto*
have $\text{adjacent } (\text{last } (\text{rotate1 } x)) (\text{hd } (\text{rotate1 } x))$
proof –
have $x \neq []$ **using** $\langle \text{length } x = p \rangle \langle \text{prime } p \rangle$ **by** *auto*
hence $\text{adjacent } (\text{last } (\text{rotate1 } x)) (\text{hd } (\text{rotate1 } x)) = \text{adjacent } (\text{hd } x) (\text{hd } (\text{tl } x))$
(tl x)
by (*metis* $\langle \text{adjacent } (\text{last } x) (\text{hd } x) \rangle$ *adjacent-no-loop append-Nil*
list.sel(1,3)
 $\text{hd-append2 last-snoc list.exhaust rotate1-hd-tl}$)
also have $\dots = \text{True}$ **using** $\langle \text{adj-path } (\text{hd } x) (\text{tl } x) \rangle$
using $\langle \text{adjacent } (\text{last } x) (\text{hd } x) \rangle \langle x \neq [] \rangle$
by (*metis* $\text{adj-path.simps}(2)$ *adjacent-no-loop append1-eq-conv*
append-Nil
 $\text{append-butlast-last-id list.sel(1,3) list.exhaust}$)
finally show *?thesis* **by** *auto*
qed
moreover have $\text{adj-path } (\text{hd } (\text{rotate1 } x)) (\text{tl } (\text{rotate1 } x))$
proof –
have $x \neq []$ **using** $\langle \text{length } x = p \rangle \langle \text{prime } p \rangle$ **by** *auto*
then obtain $y \text{ ys}$ **where** $y = \text{hd } x \text{ ys} = \text{tl } x$ **by** *auto*
hence $\text{adj-path } y \text{ ys}$ **and** $\text{adjacent } (\text{last } \text{ys}) y$ **and** $\text{ys} \neq []$
by (*metis* $\langle \text{adj-path } (\text{hd } x) (\text{tl } x) \rangle$, *metis* $\langle \text{adjacent } (\text{last } x) (\text{hd } x) \rangle \langle y$
 $= \text{hd } x \rangle$
 $\langle \text{ys} = \text{tl } x \rangle$ *adjacent-no-loop list.sel(1,3) last.simps last-tl list.exhaust*
 $, \text{metis } \langle \text{adjacent } (\text{last } x) (\text{hd } x) \rangle \langle x \neq [] \rangle \langle \text{ys} = \text{tl } x \rangle$ *adjacent-no-loop*
list.sel(1,3)
 $\text{last-ConsL neq-Nil-conv}$)
hence $\text{adj-path } (\text{hd } (\text{rotate1 } x)) (\text{tl } (\text{rotate1 } x))$
 $= \text{adj-path } (\text{hd } (\text{ys}@[y])) (\text{tl } (\text{ys}@[y]))$
using $\langle x \neq [] \rangle \langle y = \text{hd } x \rangle \langle \text{ys} = \text{tl } x \rangle$ **by** (*metis* *rotate1-hd-tl*)
also have $\dots = \text{adj-path } (\text{hd } \text{ys}) ((\text{tl } \text{ys})@[y])$
by (*metis* $\langle \text{ys} \neq [] \rangle$ *hd-append tl-append2*)
also have $\dots = \text{True}$
using *adj-path-app[OF* $\langle \text{adj-path } y \text{ ys} \rangle \langle \text{ys} \neq [] \rangle \langle \text{adjacent } (\text{last } \text{ys}) y \rangle$
 $\langle \text{ys} \neq [] \rangle$
by (*metis* *adj-path.simps(2)* *append-Cons list.sel(1,3) list.exhaust*)
finally show *?thesis* **by** *auto*
qed
moreover have $\text{length } (\text{rotate1 } x) = p$ **using** $\langle \text{length } x = p \rangle$ **by** *auto*
ultimately show $\text{rotate1 } x \in C (p - (1 :: \text{nat}))$ **using** $C \text{ p-minus-1}$ **by** *auto*
qed
have $\text{closure} : \bigwedge n x. x \in C (p - (1 :: \text{nat})) \implies \text{rotate } n x \in C (p - (1 :: \text{nat}))$
proof –
fix $n x$ **assume** $x \in C (p - (1 :: \text{nat}))$
thus $\text{rotate } n x \in C (p - (1 :: \text{nat}))$
by (*induct n, auto, metis One-nat-def closure1*)
qed


```

obtain  $r$  where  $r:r=\{(x,y). x \in C (p-(1::nat)) \wedge (\exists n < p. \text{rotate } n \ x=y)\}$  by
auto
have  $\bigwedge x. x \in C (p-(1::nat)) \implies p \text{ dvd card } \{y. (\exists n < p. \text{rotate } n \ x=y)\}$ 
proof -
  fix  $x$  assume  $x \in C (p-(1::nat))$ 
  hence  $\text{length } x=p$  using  $C \text{ p-minus-1}$  by auto
  have  $\{y. (\exists n < p. \text{rotate } n \ x=y)\} = (\lambda n. \text{rotate } n \ x)^{\{0..<p\}}$  by auto
  moreover have  $\bigwedge n1 \ n2. n1 \in \{0..<p\} \implies n2 \in \{0..<p\} \implies n1 \neq n2 \implies$ 
rotate  $n1 \ x \neq \text{rotate } n2 \ x$ 
proof
  fix  $n1 \ n2$  assume  $n1 \in \{0..<p\} \ n2 \in \{0..<p\} \ n1 \neq n2$  rotate  $n1 \ x$ 
  = rotate  $n2 \ x$ 
  { fix  $n1 \ n2$ 
  assume  $n1 \in \{0..<p\} \ n2 \in \{0..<p\} \ \text{rotate } n1 \ x = \text{rotate } n2 \ x \ n1 > n2$ 
  obtain  $s::nat$  where  $s*(n1-n2) \bmod p=1 \ s>0$ 
  proof -
    have  $n1-n2>0$  and  $n1-n2<p$ 
    using  $\langle n1 \in \{0..<p\} \rangle \langle n2 \in \{0..<p\} \rangle \langle n1 > n2 \rangle$  by auto
    with  $\langle \text{prime } p \rangle$  have  $\text{coprime } (n1 - n2) \ p$ 
    by (simp  $\text{add: prime-nat-iff'' coprime-commute [of } p]$ )
    then have  $\exists x. [(n1 - n2) * x = 1] \pmod p$ 
    by (simp  $\text{add: cong-solve-coprime-nat}$ )
    then obtain  $s$  where  $s * (n1 - n2) \bmod p = 1$ 
    using  $\langle \text{prime } p \rangle \text{prime-gt-1-nat [of } p]$ 
    by (auto simp add: cong-def ac-simps)
    moreover hence  $s>0$  by (metis mod-0 mult-0 neq0-conv)
  }
zero-neq-one)
  ultimately show ?thesis using that by auto
qed
have rotate  $(s*n1) \ x = \text{rotate } (s*n2) \ x$ 
using  $\langle \text{rotate } n1 \ x = \text{rotate } n2 \ x \rangle$ 
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  $\langle s*(n1-n2) \bmod p=1 \rangle \langle \text{length } x=p \rangle$ 
by (metis rotate-conv-mod)
hence rotate1  $x=x$  by auto
have  $\text{hd } x = \text{hd } (\text{tl } x)$  using  $\langle \text{prime } p \rangle \langle \text{length } x=p \rangle$ 
proof -
have  $\text{length } x \geq 2$  using  $\langle \text{prime } p \rangle \langle \text{length } x=p \rangle$  using prime-ge-2-nat
by blast
hence  $\text{length } (\text{tl } x) \geq 1$  by force
hence  $x \neq []$  and  $\text{tl } x \neq []$  by auto+
hence  $x = (\text{hd } x) \# (\text{hd } (\text{tl } x)) \# (\text{tl } (\text{tl } x))$  using hd-Cons-tl by auto
hence  $(\text{hd } (\text{tl } x)) \# (\text{tl } (\text{tl } x)) @ [\text{hd } x] = (\text{hd } x) \# (\text{hd } (\text{tl } x)) \# (\text{tl } (\text{tl } x))$ 
using  $\langle \text{rotate1 } x = x \rangle$  by (metis Cons-eq-appendI rotate1.simps(2))

```

```

      thus ?thesis by auto
    qed
  moreover have  $hd\ x \neq hd\ (tl\ x)$ 
  proof -
    have  $adj\_path\ (hd\ x)\ (tl\ x)$  using  $\langle x \in C\ (p - (1 :: nat)) \rangle C$  by auto
    moreover have  $length\ x \geq 2$  using  $\langle prime\ p \rangle \langle length\ x = p \rangle$  using
prime-ge-2-nat by blast
    hence  $length\ (tl\ x) \geq 1$  by force
    hence  $tl\ x \neq []$  by force
    ultimately have  $adjacent\ (hd\ x)\ (hd\ (tl\ x))$ 
      by (metis  $adj\_path.simps(2)$   $list.sel(1)$   $list.exhaust$ )
    thus ?thesis by (metis  $adjacent-no-loop$ )
  qed
  ultimately have  $False$  by auto }
  thus  $False$ 
  by (metis  $\langle n1 \in \{0..<p\} \rangle \langle n1 \neq n2 \rangle \langle n2 \in \{0..<p\} \rangle \langle rotate\ n1\ x =$ 
rotate  $n2\ x \rangle$ 
less-linear)
  qed
  hence  $inj\_on\ (\lambda n. rotate\ n\ x)\ \{0..<p\}$  unfolding  $inj\_on-def$  by fast
  ultimately have  $card\ \{y. (\exists n < p. rotate\ n\ x = y)\} = card\ \{0..<p\}$  by (metis
card-image)
  hence  $card\ \{y. (\exists n < p. rotate\ n\ x = y)\} = p$  by auto
  thus  $p\ dvd\ card\ \{y. (\exists n < p. rotate\ n\ x = y)\}$  by auto
  qed
  hence  $\forall X \in C\ (p - (1 :: nat))\ //\ r. p\ dvd\ card\ X$  unfolding  $quotient-def\ Im-$ 
age-def  $r$  by auto
  moreover have  $refl\_on\ (C\ (p - 1))\ r$ 
  proof -
    have  $r \subseteq C\ (p - 1) \times C\ (p - 1)$ 
    proof
      fix  $x$  assume  $x \in r$ 
      hence  $fst\ x \in C\ (p - 1)$  and  $\exists n. snd\ x = rotate\ n\ (fst\ x)$  using  $r$  by
auto
      moreover then obtain  $n$  where  $snd\ x = rotate\ n\ (fst\ x)$  by auto
      ultimately have  $snd\ x \in C\ (p - 1)$  using  $closure$  by auto
      moreover have  $x = (fst\ x, snd\ x)$  using  $\langle x \in r \rangle r$  by auto
      ultimately show  $x \in C\ (p - 1) \times C\ (p - 1)$  using  $\langle fst\ x \in C\ (p -$ 
1) \rangle
      by (metis  $SigmaI$ )
    qed
  moreover have  $\forall x \in C\ (p - 1). (x, x) \in r$ 
  proof
    fix  $x$  assume  $x \in C\ (p - 1)$ 
    hence  $rotate\ 0\ x \in C\ (p - 1)$  using  $closure$  by auto
    moreover have  $0 < p$  using  $\langle prime\ p \rangle$  by (auto intro:  $prime-gt-0-nat$ )
    ultimately have  $(x, rotate\ 0\ x) \in r$  using  $\langle x \in C\ (p - 1) \rangle r$  by auto
    moreover have  $rotate\ 0\ x = x$  by auto
    ultimately show  $(x, x) \in r$  by auto
  qed

```

qed
ultimately show *?thesis* **using** *refl-on-def* **by** *auto*
qed
moreover have *sym r unfolding sym-def*
proof (*rule,rule,rule*)
fix *x y* **assume** $(x, y) \in r$
hence $x \in C(p - 1)$ **using** *r* **by** *auto*
hence $\text{length } x = p$ **using** *C p-minus-1* **by** *auto*
obtain *n* **where** $n < p$ $\text{rotate } n \ x = y$ **using** $\langle (x, y) \in r \rangle$ *r* **by** *auto*
hence $y \in C(p - 1)$ **using** *closure[OF $\langle x \in C(p - 1) \rangle$]* **by** *auto*
have $n = 0 \implies (y, x) \in r$
proof –
assume $n = 0$
hence $x = y$ **using** $\langle \text{rotate } n \ x = y \rangle$ **by** *auto*
thus $(y, x) \in r$ **using** $\langle \text{refl-on } (C(p - 1)) \ r \rangle$ $\langle y \in C(p - 1) \rangle$ *refl-on-def*
by *fast*
qed
moreover have $n \neq 0 \implies (y, x) \in r$
proof –
assume $n \neq 0$
have $\text{rotate } (p - n) \ y = x$
proof –
have $\text{rotate } (p - n) \ y = \text{rotate } (p - n) \ (\text{rotate } n \ x)$
using $\langle \text{rotate } n \ x = y \rangle$ **by** *auto*
also have $\text{rotate } (p - n) \ (\text{rotate } n \ x) = \text{rotate } (p - n + n) \ x$
using *rotate-rotate* **by** *auto*
also have $\dots = \text{rotate } p \ x$ **using** $\langle n < p \rangle$ **by** *auto*
also have $\dots = \text{rotate } 0 \ x$ **using** $\langle \text{length } x = p \rangle$ **by** *auto*
also have $\dots = x$ **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** *r* $\langle y \in C(p - 1) \rangle$ **by** *auto*
qed
ultimately show $(y, x) \in r$ **by** *auto*
qed
moreover have *trans r unfolding trans-def*
proof (*rule,rule,rule,rule,rule*)
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** *C p-minus-1* **by** *auto*
obtain *n1 n2* **where** $n1 < p$ $n2 < p$ $y = \text{rotate } n1 \ x$ $z = \text{rotate } n2 \ y$
using *r* $\langle (x, y) \in r \rangle$ $\langle (y, z) \in r \rangle$ **by** *auto*
hence $z = \text{rotate } (n2 + n1) \ x$ **by** (*metis rotate-rotate*)
hence $z = \text{rotate } ((n2 + n1) \bmod p) \ x$ **using** $\langle \text{length } x = p \rangle$ **by** (*metis rotate-conv-mod*)
moreover have $(n2 + n1) \bmod p < p$ **by** (*metis $\langle \text{prime } p \rangle \text{ mod-less-divisor prime-gt-0-nat}$*)
ultimately show $(x, z) \in r$ **using** $\langle x \in C(p - 1) \rangle$ *r* **by** *auto*

```

qed
moreover have finite (C (p - 1))
  by (metis ‹card (C (p - 1)) mod p = 1› card-eq-0-iff mod-0 zero-neq-one)
  ultimately have p dvd card (C (p-(1::nat))) using equiv-imp-dvd-card
equiv-def by fast
  thus card (C (p-(1::nat))) mod p=0 by (metis dvd-eq-mod-eq-0)
qed
ultimately show False by auto
qed

```

theorem (in valid-unSimpGraph) friendship-thm:

assumes friend-assm: $\bigwedge v u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. \text{adjacent } v \ n \wedge \text{adjacent } u \ n$

and finite V

shows $\exists v. \forall n \in V. n \neq v \longrightarrow \text{adjacent } v \ n$

proof –

have card V=0 \implies ?thesis

using ‹finite V›

by (metis all-not-in-conv card-seteq empty-subsetI le0)

moreover have card V=1 \implies ?thesis

proof –

assume card V=1

then obtain v **where** V={v}

using card-eq-SucD[of V 0] **by auto**

hence $\forall n \in V. n=v$ **by auto**

thus $\exists v. \forall n \in V. n \neq v \longrightarrow \text{adjacent } v \ n$ **by auto**

qed

moreover have card V \geq 2 \implies ?thesis

proof –

assume card V \geq 2

hence $\exists v \in V. \text{degree } v \ G = 2$

using exist-degree-two[OF friend-assm] ‹finite V› **by auto**

thus ?thesis

using degree-two-windmill[OF friend-assm] ‹card V \geq 2› ‹finite V› **by auto**

qed

ultimately show ?thesis **by force**

qed

end

References

- [1] J. Q. Longyear and T. Parsons. The friendship theorem. *Indagationes Mathematicae (Proceedings)*, 75(3):257 – 262, 1972.
- [2] F. Martin. The Seven Bridges of Königsberg. <http://www.ugr.es/~fmartin/gi/bridges.pdf>.
- [3] G. B. Mertzios and W. Unger. The friendship problem on graphs. 2008.

- [4] B. Nordhoff and P. Lammich. Dijkstra's shortest path algorithm. *Archive of Formal Proofs*, June 2012. http://isa-afp.org/entries/Dijkstra_Shortest_Path.shtml, Formal proof development.