

The Königsberg Bridge Problem and the Friendship Theorem

Wenda Li

May 14, 2024

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	7
4 Adjacent nodes	9
5 Undirected simple graph	10
6 Definition of Eulerian trails and circuits	10
7 Necessary conditions for Eulerian trails and circuits	11
8 Specific case of the Königsberg Bridge Problem	11
9 Sufficient conditions for Eulerian trails and circuits	12
10 Common steps	13

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* $v \ e \ v' \ g = \text{delete-edge } v' \ e \ v \ (\text{delete-edge } v \ e \ v' \ g)$
 $\langle \text{proof} \rangle$

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

<proof>

lemma *del-unEdge-valid*[simp]: **assumes** *valid-unMultigraph g*
shows *valid-unMultigraph (del-unEdge v e v' g)*
<proof>

lemma *set-compre-diff*: $\{x \in A - B. P x\} = \{x \in A. P x\} - \{x \in B. P x\}$ *<proof>*
lemma *set-compre-subset*: $B \subseteq A \implies \{x \in B. P x\} \subseteq \{x \in A. P x\}$ *<proof>*

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>

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>

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

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

lemma *del-unEdge-com*: *del-unEdge v w v' (del-unEdge n e n' g)*
= del-unEdge n e n' (del-unEdge v w v' g)
<proof>

lemma *rem-unPath-com*: *rem-unPath ps (del-unEdge v w v' g)*
= del-unEdge v w v' (rem-unPath ps g)
<proof>

lemma *rem-unPath-valid*[intro]:
valid-unMultigraph g $\implies valid-unMultigraph (rem-unPath ps g)$
<proof>

lemma (**in** *valid-unMultigraph*) *degree-frame*:
assumes *finite (edges G) x $\notin \{v, v'\}$*
shows *degree x (del-unEdge v w v' G) = degree x G (is ?L=?R)*
<proof>

lemma [simp]: *rev-path [] = []* *<proof>*

lemma *rev-path-append[simp]*: $\text{rev-path } (xs@ys) = (\text{rev-path } ys) @ (\text{rev-path } xs)$
 ⟨proof⟩

lemma *rev-path-double[simp]*: $\text{rev-path}(\text{rev-path } xs)=xs$
 ⟨proof⟩

lemma *del-UnEdge-node[simp]*: $v \in \text{nodes } (\text{del-unEdge } u \ e \ u' \ G) \longleftrightarrow v \in \text{nodes } G$
 ⟨proof⟩

lemma [*intro!*]: $\text{finite } (\text{edges } G) \implies \text{finite } (\text{edges } (\text{del-unEdge } u \ e \ u' \ G))$
 ⟨proof⟩

lemma [*intro!*]: $\text{finite } (\text{nodes } G) \implies \text{finite } (\text{nodes } (\text{del-unEdge } u \ e \ u' \ G))$
 ⟨proof⟩

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

lemma *del-UnEdge-frame[intro]*:
 $x \in \text{edges } g \implies x \neq (v, e, v') \implies x \neq (v', e, v) \implies x \in \text{edges } (\text{del-unEdge } v \ e \ v' \ g)$
 ⟨proof⟩

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

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

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

lemma [*intro!*]: $\text{finite } (\text{nodes } G) \implies \text{finite } (\text{nodes } (\text{rem-unPath } ps \ G))$ ⟨proof⟩

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

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⟩

lemma *rem-unPath-graph [simp]*:
 $\text{rem-unPath } (\text{rev-path } ps) \ G = \text{rem-unPath } ps \ G$
 ⟨proof⟩

lemma *distinct-rev-path[simp]*: $\text{distinct } (\text{rev-path } ps) \longleftrightarrow \text{distinct } ps$
 ⟨proof⟩

lemma (**in** *valid-unMultigraph*) *is-path-rev*: $\text{is-path } v' \ (\text{rev-path } ps) \ v \longleftrightarrow \text{is-path } v \ ps \ v'$

<proof>

lemma (in *valid-unMultigraph*) *singleton-distinct-path* [intro]:

$(v, w, v') \in E \implies \text{is-trail } v [(v, w, v')] v'$

<proof>

lemma (in *valid-unMultigraph*) *is-trail-path*:

$\text{is-trail } v \text{ ps } v' \iff \text{is-path } v \text{ ps } v' \wedge \text{distinct ps} \wedge (\text{set ps} \cap \text{set (rev-path ps)}) = \{\}$

<proof>

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

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

<proof>

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

$\text{is-trail } v' \text{ ps } v \implies \text{is-path } v' \text{ ps } v$ *<proof>*

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

$\text{is-trail } v (p1 @ p2) v' \implies (\exists u. \text{is-trail } v \text{ p1 } u \wedge \text{is-trail } u \text{ p2 } v')$

<proof>

lemma (in *valid-unMultigraph*) *is-trail-split'*: $\text{is-trail } v (p1 @ (u, w, u') \# p2) v'$

$\implies \text{is-trail } v \text{ p1 } u \wedge (u, w, u') \in E \wedge \text{is-trail } u' \text{ p2 } v'$

<proof>

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

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

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

<proof>

lemma *distinct-path-subset*:

assumes *valid-unMultigraph* *G1* *valid-unMultigraph* *G2* $\text{edges } G1 \subseteq \text{edges } G2$
 $\text{nodes } G1 \subseteq \text{nodes } G2$

assumes *distinct-G1*: *valid-unMultigraph.is-trail* *G1* *v* *ps* *v'*

shows *valid-unMultigraph.is-trail* *G2* *v* *ps* *v'* *<proof>*

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>

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'*

<proof>

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

assumes *is-trail* $v ((v1,w,v2)\#ps)$ v'
shows $(v2,w,v1)\in\text{edges}(\text{rem-unPath } ps \ G) \longleftrightarrow (v2,w,v1)\in E$
 $\langle\text{proof}\rangle$

lemma (in *valid-unMultigraph*) *del-UnEdge-even*:
assumes $(v,w,v') \in E$ *finite* E
shows $v\in\text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G) \longleftrightarrow \text{even}(\text{degree } v \ G)$
 $\langle\text{proof}\rangle$

lemma (in *valid-unMultigraph*) *del-UnEdge-even'*:
assumes $(v,w,v') \in E$ *finite* E
shows $v'\in\text{odd-nodes-set}(\text{del-unEdge } v \ w \ v' \ G) \longleftrightarrow \text{even}(\text{degree } v' \ G)$
 $\langle\text{proof}\rangle$

lemma *del-UnEdge-even-even*:
assumes *valid-unMultigraph* G *finite*(*edges* G) *finite*(*nodes* G) $(v, w, v')\in\text{edges } G$
assumes *parity-assms*: *even* (*degree* $v \ G$) *even* (*degree* $v' \ G$)
shows $\text{num-of-odd-nodes}(\text{del-unEdge } v \ w \ v' \ G)=\text{num-of-odd-nodes } G + 2$
 $\langle\text{proof}\rangle$

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

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

lemma *del-UnEdge-odd-odd*:
assumes *valid-unMultigraph* G *finite*(*edges* G) *finite*(*nodes* G) $(v, w, v')\in\text{edges } G$
assumes *parity-assms*: *odd* (*degree* $v \ G$) *odd* (*degree* $v' \ G$)
shows $\text{num-of-odd-nodes } G=\text{num-of-odd-nodes}(\text{del-unEdge } v \ w \ v' \ G)+2$
 $\langle\text{proof}\rangle$

lemma (in *valid-unMultigraph*) *rem-UnPath-parity-v'*:
assumes *finite* E *is-trail* $v \ ps \ v'$
shows $v\neq v' \longleftrightarrow (\text{odd}(\text{degree } v'(\text{rem-unPath } ps \ G)) = \text{even}(\text{degree } v' \ G))$ $\langle\text{proof}\rangle$

lemma (in *valid-unMultigraph*) *rem-UnPath-parity-v*:
assumes *finite* E *is-trail* $v \ ps \ v'$
shows $v\neq v' \longleftrightarrow (\text{odd}(\text{degree } v(\text{rem-unPath } ps \ G)) = \text{even}(\text{degree } v \ G))$

<proof>

lemma (in *valid-unMultigraph*) *rem-UnPath-parity-others*:
assumes *finite E is-trail v ps v' n* $v, v' \notin \{v, v'\}$
shows $even (degree\ n\ (rem\ unPath\ ps\ G)) = even (degree\ n\ G)$ *<proof>*

lemma (in *valid-unMultigraph*) *rem-UnPath-even*:
assumes *finite E finite V is-trail v ps v'*
assumes *parity-assms: even (degree v' G)*
shows $num\ of\ odd\ nodes\ (rem\ unPath\ ps\ G) = num\ of\ odd\ nodes\ G$
 $+ (if\ even\ (degree\ v\ G) \wedge v \neq v' \ then\ 2\ else\ 0)$ *<proof>*

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 $num\ of\ odd\ nodes\ (rem\ unPath\ ps\ G) = num\ of\ odd\ nodes\ G$
 $+ (if\ odd\ (degree\ v\ G) \wedge v \neq v' \ then\ -2\ else\ 0)$ *<proof>*

lemma (in *valid-unMultigraph*) *rem-UnPath-cycle*:
assumes *finite E finite V is-trail v ps v' v=v'*
shows $num\ of\ odd\ nodes\ (rem\ unPath\ ps\ G) = num\ of\ odd\ nodes\ G$ (is ?L=?R)
<proof>

3 Connectivity

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

lemma (in *valid-unMultigraph*) $connected \implies \forall v \in V. \forall v' \in V. v \neq v' \longrightarrow (\exists ps. is\ trail\ v\ ps\ v')$
<proof>

lemma (in *valid-unMultigraph*) *no-rep-length: is-trail v ps v' $\implies length\ ps = card (set\ ps)$*
<proof>

lemma (in *valid-unMultigraph*) *path-in-edges: is-trail v ps v' $\implies set\ ps \subseteq E$*
<proof>

lemma (in *valid-unMultigraph*) *trail-bound*:
assumes *finite E is-trail v ps v'*
shows $length\ ps \leq card\ E$
<proof>

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 $\exists v. \exists \text{max-path. is-trail } v \text{ max-path } n \wedge$
 $(\forall v'. \forall e \in E. \neg \text{is-trail } v' (e \# \text{max-path}) n)$
⟨*proof*⟩

lemma *even-card'*:
assumes *even(card A) x ∈ A*
shows $\exists y \in A. y \neq x$
⟨*proof*⟩

lemma *odd-card*:
assumes *finite A odd(card A)*
shows $\exists x. x \in A$
⟨*proof*⟩

lemma (in *valid-unMultigraph*) *extend-distinct-path*:
assumes *finite E is-trail v' ps v*
assumes *parity-assms:(even (degree v' G) ∧ v' ≠ v) ∨ (odd (degree v' G) ∧ v' = v)*
shows $\exists e v1. \text{is-trail } v1 (e \# ps) v$
⟨*proof*⟩

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

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

replace-by-UnPath [] - - = [] |
replace-by-UnPath (x # xs) (v, e, v') ps =
(if x = (v, e, v') then ps @ *replace-by-UnPath* xs (v, e, v') ps
else if x = (v', e, v) then (rev-path ps) @ *replace-by-UnPath* xs (v, e, v') ps
else x # *replace-by-UnPath* xs (v, e, v') ps)

lemma (in *valid-unMultigraph*) *del-unEdge-connectivity*:
assumes *connected ∃ ps. valid-graph.is-path (del-unEdge v e v' G) v ps v'*
shows *valid-unMultigraph.connected (del-unEdge v e v' G)*
⟨*proof*⟩

lemma (in *valid-unMultigraph*) *path-between-odds*:
assumes *odd(degree v G) odd(degree v' G) finite E v ≠ v' num-of-odd-nodes G = 2*
shows $\exists ps. \text{is-trail } v \text{ ps } v'$
⟨*proof*⟩

lemma (in *valid-unMultigraph*) *del-unEdge-even-connectivity*:
assumes *finite E finite V connected ∃ n ∈ V. even(degree n G) (v, e, v') ∈ E*
shows *valid-unMultigraph.connected (del-unEdge v e v' G)*
⟨*proof*⟩

lemma (in *valid-graph*) *path-end:ps ≠ [] ⇒ is-path v ps v' ⇒ v' = snd (snd (last*

ps))
 ⟨proof⟩

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

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

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

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

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

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

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

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

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

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

and *valid-unMultigraph* $G1$

and *valid-unMultigraph* $G2$

and *valid-unMultigraph.connected* $G1$

and *valid-unMultigraph.connected* $G2$

⟨proof⟩

lemma *sub-graph-degree-frame*:

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

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

⟨proof⟩

lemma *odd-nodes-no-edge[simp]*: *finite* (*nodes* g) $\implies \text{num-of-odd-nodes } (g \setminus \text{edges} := \{\}) = 0$

⟨proof⟩

4 Adjacent nodes

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

adjacent $v v' \equiv \exists w. (v, w, v') \in E$

lemma (in *valid-unMultigraph*) *adjacent-sym*: *adjacent* $v v' \longleftrightarrow \text{adjacent } v' v$

⟨proof⟩

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

⟨proof⟩

lemma (in *valid-unMultigraph*) *adjacent-V[simp]*:

assumes *adjacent* $v v'$

shows $v \in V v' \in V$

⟨proof⟩

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

5 Undirected simple graph

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

lemma (in *valid-unSimpGraph*) *finV-to-finE[simp]*:
assumes *finite V*
shows *finite E*
 <proof>

lemma *del-unEdge-valid'[simp]:valid-unSimpGraph G* \implies
 $valid-unSimpGraph\ (del-unEdge\ v\ w\ u\ G)$
 <proof>

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>

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

end

theory *KoenigsbergBridge* **imports** *MoreGraph*
begin

6 Definition of Eulerian trails and circuits

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

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

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*⟩

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*⟩

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*⟩

8 Specific case of the Konigsberg Bridge Problem

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

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

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

instantiation *kon-node* :: *enum*

begin

definition [*simp*]: *enum-class.enum* = [*a*,*b*,*c*,*d*]

definition [*simp*]: *enum-class.enum-all* *P* $\longleftrightarrow P \ a \ \wedge \ P \ b \ \wedge \ P \ c \ \wedge \ P \ d$

definition [*simp*]: *enum-class.enum-ex* *P* $\longleftrightarrow P \ a \ \vee \ P \ b \ \vee \ P \ c \ \vee \ P \ d$

instance ⟨*proof*⟩

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* $\longleftrightarrow P \ ab1 \ \wedge \ P \ ab2 \ \wedge \ P \ ac1 \ \wedge \ P \ ac2$

$\wedge P\ ad1 \ \wedge P\ bd1$
 $\wedge P\ cd1$

definition $[simp]:enum-class.enum-ex\ P \longleftrightarrow P\ ab1 \vee P\ ab2 \vee P\ ac1 \vee P\ ac2$
 $\vee P\ ad1 \vee P\ bd1$
 $\vee P\ cd1$

instance $\langle proof \rangle$
end

interpretation $kon-graph: valid-unMultigraph\ kon-graph$
 $\langle proof \rangle$

theorem $\neg kon-graph.is-Eulerian-trail\ v1\ p\ v2$
 $\langle proof \rangle$

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 $is-Eulerian-trail\ v0\ ((v0,w,v1)\#ps)\ v2$
 $\langle proof \rangle$

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 $is-Eulerian-trail\ v1\ (ps@[v2,w,v3])\ v3$
 $\langle proof \rangle$

lemma $eulerian-split$:

assumes $nodes\ G1 \cap nodes\ G2 = \{\}$ $edges\ G1 \cap edges\ G2 = \{\}$

$valid-unMultigraph\ G1\ valid-unMultigraph\ G2$
 $valid-unMultigraph.is-Eulerian-trail\ G1\ v1\ ps1\ v1'$
 $valid-unMultigraph.is-Eulerian-trail\ G2\ v2\ ps2\ v2'$

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

lemma (in $valid-unMultigraph$) $eulerian-sufficient$:

assumes $finite\ V\ finite\ E\ connected\ V \neq \{\}$

shows $num-of-odd-nodes\ G = 2 \implies$

$(\exists v \in V. \exists v' \in V. \exists ps. odd(degree\ v\ G) \wedge odd(degree\ v'\ G) \wedge (v \neq v') \wedge is-Eulerian-trail\ v\ ps\ v')$

and $num-of-odd-nodes\ G = 0 \implies (\forall v \in V. \exists ps. is-Eulerian-circuit\ v\ ps\ v)$

<proof>
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'* ≡ $v \in V \wedge v' \in V \wedge v \neq v' \wedge \neg \text{adjacent } v v'$

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 $\neg (\exists v1 v2 v3 v4. v2 \neq v4 \wedge v1 \neq v3 \wedge \text{adjacent } v1 v2 \wedge \text{adjacent } v2 v3 \wedge$
 $\text{adjacent } v3 v4$
 $\wedge \text{adjacent } v4 v1)$
 <proof>

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*) <proof>

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>

lemma (in *valid-unSimpGraph*) *degree-two-windmill*:
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* **and** *card* *V* ≥ 2
shows $(\exists v \in V. \text{degree } v G = 2) \iff (\exists v. \forall n \in V. n \neq v \longrightarrow \text{adjacent } v n)$
 <proof>

lemma (in *valid-unSimpGraph*) *regular*:
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* **and** *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>

11 Exclusive steps for combinatorial proofs

fun (in *valid-unSimpGraph*) *adj-path*:: '*v* ⇒ '*v* list ⇒ bool **where**

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

lemma (in *valid-unSimpGraph*) *adj-path-butlast*:
 $adj\text{-path } v\ ps \implies adj\text{-path } v\ (butlast\ ps)$
 <proof>

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

lemma (in *valid-unSimpGraph*) *adj-path-V'*:
 $adj\text{-path } v\ ps \implies v \in V$
 <proof>

lemma (in *valid-unSimpGraph*) *adj-path-app*:
 $adj\text{-path } v\ ps \implies ps \neq [] \implies adjacent\ (last\ ps)\ u \implies adj\text{-path } v\ (ps @ [u])$
 <proof>

lemma (in *valid-unSimpGraph*) *adj-path-app'*:
 $adj\text{-path } v\ (ps @ [q]) \implies ps \neq [] \implies adjacent\ (last\ ps)\ q$
 <proof>

lemma *card-partition'*:
assumes $\forall v \in A. card\ \{n. R\ v\ n\} = k\ k > 0$ *finite A*
 $\forall v1\ v2. v1 \neq v2 \longrightarrow \{n. R\ v1\ n\} \cap \{n. R\ v2\ n\} = \{\}$
shows $card\ (\bigcup v \in A. \{n. R\ v\ n\}) = k * card\ A$
 <proof>

lemma (in *valid-unSimpGraph*) *path-count*:
assumes $k\text{-adj}: \bigwedge v. v \in V \implies card\ \{n. adjacent\ v\ n\} = k$ **and** $v \in V$ **and** *finite V* **and** $k > 0$
shows $card\ \{ps. length\ ps = l \wedge adj\text{-path } v\ ps\} = k^l$
 <proof>

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>

lemma *rotate-eq*: $rotate1\ xs = rotate1\ ys \implies xs = ys$
 <proof>

lemma *rotate-diff*: $rotate\ m\ xs = rotate\ n\ xs \implies rotate\ (m - n)\ xs = xs$
 <proof>

lemma (in *valid-unSimpGraph*) *exist-degree-two*:
assumes *friend-asm*: $\bigwedge v u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. \text{adjacent } v \ n \wedge \text{adjacent } u \ n$
and *finite E* **and** *finite V* **and** *card V* ≥ 2
shows $\exists v \in V. \text{degree } v \ G = 2$
<proof>

theorem (in *valid-unSimpGraph*) *friendship-thm*:
assumes *friend-asm*: $\bigwedge v u. v \in V \implies u \in V \implies v \neq u \implies \exists! n. \text{adjacent } v \ n \wedge \text{adjacent } u \ n$
and *finite V*
shows $\exists v. \forall n \in V. n \neq v \longrightarrow \text{adjacent } v \ n$
<proof>

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.