Menger's Theorem

Christoph Dittmann
isabelle@christoph-d.de

March 17, 2025

We present a formalization of Menger's Theorem for directed and undirected graphs in Isabelle/HOL. This well-known result shows that if two non-adjacent distinct vertices u, v in a directed graph have no separator smaller than n, then there exist n internally vertex-disjoint paths from u to v.

The version for undirected graphs follows immediately because undirected graphs are a special case of directed graphs.

Contents

1 Introduction **3**

2 Relation to Min-Cut Max-Flow **3**

3 Helpers **3**

4 Graphs **4**
4.1 Walks 5
4.2 Paths 5
4.3 The Set of All Paths 6
4.4 Edges of Walks 7
4.5 The First Edge of a Walk 8
4.6 Distance 9
4.7 Subgraphs 9
4.8 Two Distinguished Distinct Non-adjacent Vertices. 10
4.9 Undirected Graphs 10

5 Separations **11**

6 Internally Vertex-Disjoint Paths **11**
6.1 Basic Properties 12
6.2 Second Vertices 13

7 One More Path **13**
7.1 Characterizing the New Path 14

7.2 The Last Vertex of the New Path 14
7.3 Removing the Last Vertex 15
7.4 A New Path Following the Other Paths 15

8 Induction of Menger's Theorem **16**
8.1 No Small Separations 16
8.2 Choosing Paths Avoiding *new_last* 16
8.3 Finding a Path Avoiding Q 17
8.4 Decomposing P_k 18

9 The case $y = new\_last$ **18**

10 The case $y \neq new\_last$ **20**

11 Menger's Theorem **21**
11.1 Menger's Theorem 22
11.2 Self-contained Statement of the Main Theorem 22

Bibliography **23**

1 Introduction

Given two non-adjacent distinct vertices u, v in a finite directed graph, a *u-v-separator* is a set of vertices S with $u \notin S, v \notin S$ such that every u-v-path visits a vertex of S. Two u-v-paths are *internally vertex-disjoint* if their intersection is exactly $\{u, v\}$.

A famous classical result of graph theory relates the size of a minimum separator to the maximal number of internally vertex-disjoint paths.

Theorem 1 (Menger [Men27]) *Let u, v be two non-adjacent distinct vertices. Then the size of a minimum u-v-separator equals the maximal number of pairwise internally vertex-disjoint u-v-paths.*

This theorem has many proofs, but as far as the author is aware, there was no formalized proof. We follow a proof given by William McCuaig, who calls it "A simple proof of Menger's theorem" [McC84]. His proof is roughly one page in length. Our formalization is significantly longer than that because we had to fill in a lot of details.

Most of the work goes into showing the following theorem, which proves one direction of Theorem 1.

Theorem 2 *Let u, v be two non-adjacent distinct vertices. If every u-v-separator has size at least n, then there exists n pairwise internally vertex-disjoint u-v-paths.*

Compared to this, the other direction of Theorem 1 is easy because the existence of n internally vertex-disjoint paths implies that every separator needs to cut at least these paths, so every separator needs to have size at least n.

2 Relation to Min-Cut Max-Flow

Another famous result of graph theory is the Min-Cut Max-Flow Theorem, stating that the size of a minimum u-v-cut equals the value of a maximum u-v-flow. There exists a formalization of a very general version of this theorem for countable graphs in the Archive of Formal Proofs, written by Andreas Lochbihler [Loc16].

Technically, our version of Menger's Theorem should follow from Lochbihler's very general result. However, the author was of the opinion that a fresh formalization of Menger's Theorem was warranted given the complexity of the Min-Cut Max-Flow formalization. Our formalization is about a sixth of the size of the Min-Cut Max-Flow formalization (not counting comments). It may also be easier to grasp by readers who are unfamiliar with the intricacies of countable networks.

Let us also note that the Min-Cut Max-Flow Theorem considers *edge cuts* whereas Menger's Theorem works with *vertex cuts*. This is a minor difference because one can be reduced to the other, but it makes Menger's Theorem not a trivial corollary of the Min-Cut Max-Flow formalization.

3 Helpers

theory *Helpers* **imports** *Main* **begin**

First, we will prove a few lemmas unrelated to graphs or Menger's Theorem. These lemmas will simplify some of the other proof steps.

If two finite sets have different cardinality, then there exists an element in the larger set that is not in the smaller set.

lemma *card-finite-less-ex*:
  assumes *finite-A*: *finite A*
    and *finite-B*: *finite B*
    and *card-AB*: $card\ A < card\ B$
  shows $\exists\, b \in B.\ b \notin A$
⟨*proof*⟩

The cardinality of the union of two disjoint finite sets is the sum of their cardinalities even if we intersect everything with a fixed set X.

lemma *card-intersect-sum-disjoint*:
  assumes *finite B finite C* $A = B \cup C$ $B \cap C = \{\}$
   shows $card\ (A \cap X) = card\ (B \cap X) + card\ (C \cap X)$
  ⟨*proof*⟩

If x is in a list *xs* but is not its last element, then it is also in *butlast xs*.

lemma *set-butlast*: $[\![\ x \in set\ xs;\ x \neq last\ xs\ ]\!] \Longrightarrow x \in set\ (butlast\ xs)$
  ⟨*proof*⟩

If a property P is satisfiable and if we have a weight measure mapping into the natural numbers, then there exists an element of minimum weight satisfying P because the natural numbers are well-ordered.

lemma *arg-min-ex*:
  fixes P :: *'a ⇒ bool* **and** *weight* :: *'a ⇒ nat*
  assumes $\exists\, x.\ P\ x$
  obtains x **where** $P\ x\ \bigwedge y.\ P\ y \Longrightarrow weight\ x \leq weight\ y$
⟨*proof*⟩

end

4 Graphs

theory *Graph* **imports** *Main* **begin**

Let us now define digraphs, graphs, walks, paths, and related concepts.

'a is the vertex type.

type-synonym *'a Edge* = *'a* × *'a*
type-synonym *'a Walk* = *'a list*

record *'a Graph* =
  verts :: *'a set* (‹V_1›)
  arcs :: *'a Edge set* (‹E_1›)
abbreviation *is-arc* :: (*'a*, *'b*) *Graph-scheme* ⇒ *'a* ⇒ *'a* ⇒ *bool* (**infixl** ‹$\rightarrow_1$› *60*) **where**
  $v \rightarrow_G w \equiv (v,w) \in E_G$

We consider directed and undirected finite graphs. Our graphs do not have multi-edges.

locale *Digraph* =
  fixes *G* :: ('*a*, '*b*) *Graph-scheme* (**structure**)
  assumes *finite-vertex-set*: *finite V*
    and *valid-edge-set*: $E \subseteq V \times V$

context *Digraph* **begin**

lemma *finite-edge-set* [*simp*]: *finite E* ⟨*proof*⟩
lemma *edges-are-in-V*: **assumes** $v \rightarrow w$ **shows** $v \in V$ $w \in V$
  ⟨*proof*⟩

4.1 Walks

A walk is sequence of vertices connected by edges.

inductive *walk* :: '*a Walk* ⇒ *bool* **where**
Nil [*simp*]: *walk* []
| *Singleton* [*simp*]: $v \in V \Longrightarrow$ *walk* [*v*]
| *Cons*: $v \rightarrow w \Longrightarrow$ *walk* (*w* # *vs*) $\Longrightarrow$ *walk* (*v* # *w* # *vs*)

Show a few composition/decomposition lemmas for walks. These will greatly simplify the proofs that follow.

lemma *walk-2* [*simp*]: $v \rightarrow w \Longrightarrow$ *walk* [*v*,*w*] ⟨*proof*⟩
lemma *walk-comp*: ⟦ *walk xs*; *walk ys*; *xs* = *Nil* ∨ *ys* = *Nil* ∨ *last xs*→*hd ys* ⟧ ⟹ *walk* (*xs* @ *ys*)
  ⟨*proof*⟩
lemma *walk-tl*: *walk xs* ⟹ *walk* (*tl xs*) ⟨*proof*⟩
lemma *walk-drop*: *walk xs* ⟹ *walk* (*drop n xs*) ⟨*proof*⟩
lemma *walk-take*: *walk xs* ⟹ *walk* (*take n xs*)
  ⟨*proof*⟩
lemma *walk-decomp*: **assumes** *walk* (*xs* @ *ys*) **shows** *walk xs walk ys*
  ⟨*proof*⟩
lemma *walk-in-V*: *walk xs* ⟹ *set xs* ⊆ *V* ⟨*proof*⟩
lemma *walk-first-edge*: *walk* (*v* # *w* # *xs*) ⟹ *v*→*w* ⟨*proof*⟩
lemma *walk-first-edge'*: ⟦ *walk* (*v* # *xs*); *xs* ≠ *Nil* ⟧ ⟹ *v*→*hd xs*
  ⟨*proof*⟩
lemma *walk-middle-edge*: *walk* (*xs* @ *v* # *w* # *ys*) ⟹ *v*→*w*
  ⟨*proof*⟩
lemma *walk-last-edge*: ⟦ *walk* (*xs* @ *ys*); *xs* ≠ *Nil*; *ys* ≠ *Nil* ⟧ ⟹ *last xs*→*hd ys*
  ⟨*proof*⟩

4.2 Paths

A path is a walk without repeated vertices. This is simple enough, so most of the above lemmas transfer directly to paths.

abbreviation *path* :: '*a Walk* ⇒ *bool* **where** *path xs* ≡ *walk xs* ∧ *distinct xs*

lemma *path-singleton* [*simp*]: *v* ∈ *V* ⟹ *path* [*v*] ⟨*proof*⟩
lemma *path-2* [*simp*]: ⟦ *v*→*w*; *v* ≠ *w* ⟧ ⟹ *path* [*v*,*w*] ⟨*proof*⟩
lemma *path-cons*: ⟦ *path xs*; *xs* ≠ *Nil*; *v*→*hd xs*; *v* ∉ *set xs* ⟧ ⟹ *path* (*v* # *xs*)

⟨*proof*⟩
lemma *path-comp*: $\llbracket$ *walk xs*; *walk ys*; $xs = Nil \lor ys = Nil \lor last\ xs {\rightarrow} hd\ ys$; *distinct* (*xs* @ *ys*) $\rrbracket$
$\Longrightarrow$ *path* (*xs* @ *ys*) ⟨*proof*⟩
lemma *path-tl*: *path xs* $\Longrightarrow$ *path* (*tl xs*) ⟨*proof*⟩
lemma *path-drop*: *path xs* $\Longrightarrow$ *path* (*drop n xs*) ⟨*proof*⟩
lemma *path-take*: *path xs* $\Longrightarrow$ *path* (*take n xs*) ⟨*proof*⟩
lemma *path-decomp*: **assumes** *path* (*xs* @ *ys*) **shows** *path xs path ys*
⟨*proof*⟩
lemma *path-decomp'*: *path* (*xs* @ *x* # *ys*) $\Longrightarrow$ *path* (*xs* @ [*x*])
⟨*proof*⟩
lemma *path-in-V*: *path xs* $\Longrightarrow$ *set xs* $\subseteq$ *V* ⟨*proof*⟩
lemma *path-length*: *path xs* $\Longrightarrow$ *length xs* $\leq$ *card V*
⟨*proof*⟩
lemma *path-first-edge*: *path* (*v* # *w* # *xs*) $\Longrightarrow$ $v {\rightarrow} w$ ⟨*proof*⟩
lemma *path-first-edge'*: $\llbracket$ *path* (*v* # *xs*); $xs \neq Nil$ $\rrbracket$ $\Longrightarrow$ $v {\rightarrow} hd\ xs$ ⟨*proof*⟩
lemma *path-middle-edge*: *path* (*xs* @ *v* # *w* # *ys*) $\Longrightarrow$ $v \rightarrow w$ ⟨*proof*⟩
lemma *path-first-vertex*: *path* (*x* # *xs*) $\Longrightarrow$ $x \notin set\ xs$ ⟨*proof*⟩
lemma *path-disjoint*: $\llbracket$ *path* (*xs* @ *ys*); $xs \neq Nil$; $x \in set\ xs$ $\rrbracket$ $\Longrightarrow$ $x \notin set\ ys$ ⟨*proof*⟩

4.3 The Set of All Paths

definition *all-paths* **where** *all-paths* $\equiv$ { *xs* | *xs*. *path xs* }

Because paths have no repeated vertices, every graph has at most finitely many distinct paths. This will be useful later to easily derive that any set of paths is finite.

lemma *finitely-many-paths*: *finite all-paths* ⟨*proof*⟩

end — context Digraph

We introduce shorthand notation for a path connecting two vertices.

definition *path-from-to* :: (*'a*, *'b*) *Graph-scheme* $\Rightarrow$ *'a* $\Rightarrow$ *'a Walk* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
(‹- ⇝-⇝ı -› [*71*, *71*, *71*] *70*) **where**
path-from-to G v xs w $\equiv$ *Digraph.path G xs* $\land$ $xs \neq Nil$ $\land$ $hd\ xs = v$ $\land$ $last\ xs = w$

context *Digraph* **begin**

lemma *path-from-toI* [*intro*]: $\llbracket$ *path xs*; $xs \neq Nil$; $hd\ xs = v$; $last\ xs = w$ $\rrbracket$ $\Longrightarrow$ *v* ⇝*xs*⇝ *w*
and *path-from-toE* [*dest*]: *v* ⇝*xs*⇝ *w* $\Longrightarrow$ *path xs* $\land$ $xs \neq Nil$ $\land$ $hd\ xs = v$ $\land$ $last\ xs = w$
⟨*proof*⟩

lemma *path-from-to-ends*: *v* ⇝(*xs* @ *w* # *ys*)⇝ *w* $\Longrightarrow$ *ys* = *Nil*
⟨*proof*⟩

lemma *path-from-to-combine*:
assumes *v* ⇝(*xs* @ *x* # *xs'*)⇝ *w v'* ⇝(*ys* @ *x* # *ys'*)⇝ *w' set xs* $\cap$ *set ys'* = {}
shows *v* ⇝(*xs* @ *x* # *ys'*) ⇝ *w'*
⟨*proof*⟩

lemma *path-from-to-first*: *v* ⇝*xs*⇝ *w* $\Longrightarrow$ $v \notin set$ (*tl xs*)
⟨*proof*⟩

lemma *path-from-to-first'*: $v \rightsquigarrow(xs\ @\ x\ \#\ xs')\rightsquigarrow\ w \Longrightarrow v \notin set\ xs'$
⟨*proof*⟩

lemma *path-from-to-last*: $v \rightsquigarrow xs \rightsquigarrow\ w \Longrightarrow w \notin set\ (butlast\ xs)$
⟨*proof*⟩

lemma *path-from-to-last'*: $v \rightsquigarrow(xs\ @\ x\ \#\ xs')\rightsquigarrow\ w \Longrightarrow w \notin set\ xs$
⟨*proof*⟩

Every walk contains a path connecting the same vertices.

lemma *walk-to-path*:
assumes $walk\ xs\ xs \neq Nil\ hd\ xs = v\ last\ xs = w$
shows $\exists\, ys.\ v \rightsquigarrow ys \rightsquigarrow\ w \wedge set\ ys \subseteq set\ xs$
⟨*proof*⟩

4.4 Edges of Walks

The set of edges on a walk. Note that this is empty for walks of length 0 or 1.

definition $edges\text{-}of\text{-}walk :: {}'a\ Walk \Rightarrow {}'a\ Edge\ set$ **where**
$edges\text{-}of\text{-}walk\ xs = \{\ (v,w) \mid v\ w\ xs\text{-}pre\ xs\text{-}post.\ xs = xs\text{-}pre\ @\ v\ \#\ w\ \#\ xs\text{-}post\ \}$

lemma *edges-of-walkE*: $(v,w) \in edges\text{-}of\text{-}walk\ xs \Longrightarrow \exists\, xs\text{-}pre\ xs\text{-}post.\ xs = xs\text{-}pre\ @\ v\ \#\ w\ \#\ xs\text{-}post$
⟨*proof*⟩

lemma *edges-of-walk-in-E*: $walk\ xs \Longrightarrow edges\text{-}of\text{-}walk\ xs \subseteq E$
⟨*proof*⟩

lemma *edges-of-walk-finite*: $walk\ xs \Longrightarrow finite\ (edges\text{-}of\text{-}walk\ xs)$
⟨*proof*⟩

lemma *edges-of-walk-empty*: $edges\text{-}of\text{-}walk\ [] = \{\}\ edges\text{-}of\text{-}walk\ [v] = \{\}$
⟨*proof*⟩

lemma *edges-of-walk-2*: $edges\text{-}of\text{-}walk\ [v,w] = \{(v,w)\}$ ⟨*proof*⟩

lemma *edges-of-walk-edge*: $[\![\ walk\ xs;\ (v,w) \in edges\text{-}of\text{-}walk\ xs\ ]\!] \Longrightarrow v \rightarrow w$
⟨*proof*⟩

lemma *edges-of-walk-middle* [*simp*]: $(v,w) \in edges\text{-}of\text{-}walk\ (xs\ @\ v\ \#\ w\ \#\ xs')$
⟨*proof*⟩

lemma *edges-of-comp1*: $edges\text{-}of\text{-}walk\ xs \subseteq edges\text{-}of\text{-}walk\ (xs\ @\ ys)$
⟨*proof*⟩
lemma *edges-of-comp2*: $edges\text{-}of\text{-}walk\ ys \subseteq edges\text{-}of\text{-}walk\ (xs\ @\ ys)$ ⟨*proof*⟩

lemma *walk-edges-decomp-simple*:
$edges\text{-}of\text{-}walk\ (v\ \#\ w\ \#\ xs) = \{(v,w)\} \cup edges\text{-}of\text{-}walk\ (w\ \#\ xs)$ (**is** *?A = ?B*)
⟨*proof*⟩

lemma *walk-edges-decomp*:
$edges\text{-}of\text{-}walk\ (xs\ @\ x\ \#\ xs') = edges\text{-}of\text{-}walk\ (xs\ @\ [x]) \cup edges\text{-}of\text{-}walk\ (x\ \#\ xs')$

⟨*proof*⟩

lemma *walk-edges-decomp′*:
edges-of-walk (*xs* @ *v* # *w* # *xs′*) = *edges-of-walk* (*xs* @ [*v*]) ∪ {(*v*,*w*)} ∪ *edges-of-walk* (*w* # *xs′*)
⟨*proof*⟩

lemma *walk-edges-vertices*: **assumes** (*v*, *w*) ∈ *edges-of-walk xs* **shows** *v* ∈ *set xs w* ∈ *set xs*
⟨*proof*⟩

lemma *walk-edges-subset*:
assumes *edges-subsets*: *edges-of-walk xs* ⊆ *edges-of-walk ys*
and *non-trivial*: *tl xs* ≠ *Nil*
shows *set xs* ⊆ *set ys*
⟨*proof*⟩

A path has no repeated vertices, so if we split a path at an edge we find that the two pieces do not contain this edge any more.

lemma *path-edges*:
assumes *path xs* (*v*,*w*) ∈ *edges-of-walk xs*
shows ∃*xs-pre xs-post*. *xs* = *xs-pre* @ *v* # *w* # *xs-post*
∧ (*v*,*w*) ∉ *edges-of-walk* (*xs-pre* @ [*v*])
∧ (*v*,*w*) ∉ *edges-of-walk* (*w* # *xs-post*)
⟨*proof*⟩

lemma *path-edges-remove-prefix*:
assumes *path* (*xs* @ *x* # *xs′*)
shows *edges-of-walk* (*xs* @ [*x*]) = *edges-of-walk* (*xs* @ *x* # *xs′*) − *edges-of-walk* (*x* # *xs′*)
⟨*proof*⟩

4.5 The First Edge of a Walk

In the proof of Menger's Theorem, we will often talk about the first edge of a path. Let us define this concept.

fun *first-edge-of-walk* **where**
first-edge-of-walk (*v* # *w* # *xs*) = (*v*, *w*)
| *first-edge-of-walk* [*v*] = *undefined*
| *first-edge-of-walk* [] = *undefined*

lemma *first-edge-in-edges*: *tl xs* ≠ *Nil* ⟹ *first-edge-of-walk xs* ∈ *edges-of-walk xs*
⟨*proof*⟩

lemma *first-edge-hd-tl*: ⟦ *v* ⇝*xs*⇝ *w*; *tl xs* ≠ *Nil* ⟧ ⟹ *first-edge-of-walk xs* = (*v*, *hd* (*tl xs*))
⟨*proof*⟩

lemma *first-edge-first*:
assumes *v* ⇝*xs*⇝ *w* (*v*,*w′*) ∈ *edges-of-walk xs*
shows *first-edge-of-walk xs* = (*v*,*w′*)
⟨*proof*⟩

4.6 Distance

The distance between two vertices is the minimum length of a path. Note that this is not a symmetric function because we are on digraphs.

definition *distance* :: $'a \Rightarrow 'a \Rightarrow nat$ **where**
distance $v\ w \equiv Min\ \{\ length\ xs \mid xs.\ v{\leadsto}xs{\leadsto}w\ \}$

The *Min* operator applies only to finite sets, so let us prove that this is the case.

lemma *distance-lengths-finite*: $finite\ \{\ length\ xs \mid xs.\ v{\leadsto}xs{\leadsto}w\ \}$ $\langle proof \rangle$

If we have a concrete path from v to w, then the length of this path bounds the distance from v to w.

lemma *distance-upper-bound*: $v{\leadsto}xs{\leadsto}w \Longrightarrow distance\ v\ w \leq length\ xs$
$\langle proof \rangle$

Another characterization of *distance*: If we have a concrete minimal path from v to w, this defines the distance.

lemma *distance-witness*:
assumes *xs*: $v \leadsto xs \leadsto w$
and *xs-min*: $\bigwedge xs'.\ v \leadsto xs' \leadsto w \Longrightarrow length\ xs \leq length\ xs'$
shows $distance\ v\ w = length\ xs$
$\langle proof \rangle$

4.7 Subgraphs

We only need one kind of subgraph: The subgraph obtained by removing a single vertex.

definition *remove-vertex* :: $'a \Rightarrow ('a, 'b)$ *Graph-scheme* **where**
remove-vertex $x \equiv G(\!|\ verts := V - \{x\},\ arcs := Restr\ E\ (V - \{x\})\ |\!)$

lemma *remove-vertex-V*: $V_{remove\text{-}vertex\ x} = V - \{x\}$ $\langle proof \rangle$
lemma *remove-vertex-V'*: $V_{remove\text{-}vertex\ x} \subseteq V$ $\langle proof \rangle$
lemma *remove-vertex-E*: $E_{remove\text{-}vertex\ x} = Restr\ E\ (V - \{x\})$ $\langle proof \rangle$
lemma *remove-vertex-E'*: $v \rightarrow_{remove\text{-}vertex\ x} w \Longrightarrow v{\rightarrow}w$ $\langle proof \rangle$
lemma *remove-vertex-E''*: $[\![\ v{\rightarrow}w;\ v \neq x;\ w \neq x\ ]\!] \Longrightarrow v \rightarrow_{remove\text{-}vertex\ x} w$
$\langle proof \rangle$

Of course, this is still a digraph.

lemma *remove-vertex-Digraph*: *Digraph* (*remove-vertex* v) $\langle proof \rangle$

We are also going to need a few lemmas about how walks and paths behave when we remove a vertex.

First, if we remove a vertex that is not on a walk *xs*, then *xs* is still a walk after removing this vertex.

lemma *remove-vertex-walk*:
assumes $walk\ xs\ x \notin set\ xs$
shows *Digraph.walk* (*remove-vertex* x) *xs*
$\langle proof \rangle$

The same holds for paths.

lemma *remove-vertex-path-from-to*:
  $[\![v \leadsto xs \leadsto w;\ x \in V;\ x \notin set\ xs]\!] \Longrightarrow v \leadsto xs \leadsto_{remove\text{-}vertex\ x} w$
  ⟨*proof*⟩

Conversely, if something was a walk or a path in the subgraph, then it is also a walk or a path in the supergraph.

lemma *remove-vertex-walk-add*:
  assumes *Digraph.walk* (*remove-vertex x*) *xs*
  shows *walk xs*
⟨*proof*⟩

lemma *remove-vertex-path-from-to-add*: $v \leadsto xs \leadsto_{remove\text{-}vertex\ x} w \Longrightarrow v \leadsto xs \leadsto w$
  ⟨*proof*⟩

end — context Digraph

4.8 Two Distinguished Distinct Non-adjacent Vertices.

The setup for Menger's Theorem requires two distinguished distinct non-adjacent vertices *v0* and *v1*. Let us pin down this concept with the following locale.

locale *v0-v1-Digraph* = *Digraph* +
  fixes *v0 v1* :: *'a*
  assumes *v0-V*: $v0 \in V$ **and** *v1-V*: $v1 \in V$
    and *v0-nonadj-v1*: $\neg v0 \rightarrow v1$
    and *v0-neq-v1*: $v0 \neq v1$

The only lemma we need about *v0-v1-Digraph* for now is that it is closed under removing a vertex that is not *v0* or *v1*.

lemma (**in** *v0-v1-Digraph*) *remove-vertices-v0-v1-Digraph*:
  assumes $v \neq v0\ v \neq v1$
  shows *v0-v1-Digraph* (*remove-vertex v*) *v0 v1*
⟨*proof*⟩

4.9 Undirected Graphs

We represent undirecteded graphs as a special case of digraphs where every undirected edge is represented as an edge in both directions. We also exclude loops because loops are uncommon in undirected graphs.

As we will explain in the next paragraph, all of this has no bearing on the validity of Menger's Theorem for undirected graphs.

locale *Graph* = *Digraph* +
  assumes *undirected*: $v \rightarrow w = w \rightarrow v$
    and *no-loops*: $\neg v \rightarrow v$

We observe that this makes *Digraph* a sublocale of *Graph*, meaning that every theorem we prove for digraphs automatically holds for undirected graphs, although it may not make sense because for example "connectedness" (if we were to define it) would need different definitions for directed and undirected graphs.

Fortunately, the notions of "separator" and "internally vertex-disjoint paths" on directed graphs are the same for undirected graphs. So Menger's Theorem, when we eventually prove it in the *Digraph* locale, will apply automatically to the *Graph* locale without any additional work.

For this reason we will not use the *Graph* locale again in this proof development and it exists merely to show that undirected graphs are covered as a special case by our definitions.

end

5 Separations

theory *Separations* **imports** *Helpers Graph* **begin**

locale *Separation* = *v0-v1-Digraph* +
  fixes $S :: 'a\ set$
  assumes *S-V*: $S \subseteq V$
    and *v0-notin-S*: $v0 \notin S$
    and *v1-notin-S*: $v1 \notin S$
    and *S-separates*: $\bigwedge xs.\ v0 \leadsto xs \leadsto v1 \Longrightarrow set\ xs \cap S \neq \{\}$

lemma (**in** *Separation*) *finite-S* [*simp*]: *finite S* ⟨*proof*⟩

lemma (**in** *v0-v1-Digraph*) *subgraph-separation-extend*:
  assumes $v \neq v0\ v \neq v1\ v \in V$
    and *Separation* (*remove-vertex v*) *v0 v1 S*
  shows *Separation G v0 v1* (*insert v S*)
⟨*proof*⟩

lemma (**in** *v0-v1-Digraph*) *subgraph-separation-min-size*:
  assumes $v \neq v0\ v \neq v1\ v \in V$
    and *no-small-separation*: $\bigwedge S.\ Separation\ G\ v0\ v1\ S \Longrightarrow card\ S \geq Suc\ n$
    and *Separation* (*remove-vertex v*) *v0 v1 S*
  shows $card\ S \geq n$
  ⟨*proof*⟩

lemma (**in** *v0-v1-Digraph*) *path-exists-if-no-separation*:
  assumes $S \subseteq V\ v0 \notin S\ v1 \notin S\ \neg Separation\ G\ v0\ v1\ S$
  shows $\exists xs.\ v0 \leadsto xs \leadsto v1 \land set\ xs \cap S = \{\}$
  ⟨*proof*⟩

end

6 Internally Vertex-Disjoint Paths

theory *DisjointPaths* **imports** *Separations* **begin**

Menger's Theorem talks about internally vertex-disjoint *v0-v1*-paths. Let us define this concept.

locale *DisjointPaths* = *v0-v1-Digraph* +
  fixes $paths :: 'a\ Walk\ set$

assumes *paths*:
$\bigwedge xs.\ xs \in paths \Longrightarrow v0 \leadsto xs \leadsto v1$
and *paths-disjoint*: $\bigwedge xs\ ys\ v.$
$[\![\ xs \in paths;\ ys \in paths;\ xs \neq ys;\ v \in set\ xs;\ v \in set\ ys\ ]\!] \Longrightarrow v = v0 \lor v = v1$

6.1 Basic Properties

The empty set of paths trivially satisfies the conditions.

lemma (**in** *v0-v1-Digraph*) *DisjointPaths-empty*: *DisjointPaths G v0 v1* {}
⟨*proof*⟩

Re-adding a deleted vertex is fine.

lemma (**in** *v0-v1-Digraph*) *DisjointPaths-supergraph*:
assumes *DisjointPaths* (*remove-vertex v*) *v0 v1 paths*
shows *DisjointPaths G v0 v1 paths*
⟨*proof*⟩

context *DisjointPaths* **begin**

lemma *paths-in-all-paths*: $paths \subseteq all\text{-}paths$ ⟨*proof*⟩
lemma *finite-paths*: *finite paths*
⟨*proof*⟩

lemma *paths-edge-finite*: $finite\ (\bigcup (edges\text{-}of\text{-}walk\ `\ paths))$ ⟨*proof*⟩

lemma *paths-tl-notnil*: $xs \in paths \Longrightarrow tl\ xs \neq Nil$
⟨*proof*⟩

lemma *paths-second-in-V*: $xs \in paths \Longrightarrow hd\ (tl\ xs) \in V$
⟨*proof*⟩

lemma *paths-second-not-v0*: $xs \in paths \Longrightarrow hd\ (tl\ xs) \neq v0$
⟨*proof*⟩

lemma *paths-second-not-v1*: $xs \in paths \Longrightarrow hd\ (tl\ xs) \neq v1$
⟨*proof*⟩

lemma *paths-second-disjoint*: $[\![\ xs \in paths;\ ys \in paths;\ xs \neq ys\ ]\!] \Longrightarrow hd\ (tl\ xs) \neq hd\ (tl\ ys)$
⟨*proof*⟩

lemma *paths-edge-disjoint*:
assumes $xs \in paths\ ys \in paths\ xs \neq ys$
shows $edges\text{-}of\text{-}walk\ xs \cap edges\text{-}of\text{-}walk\ ys = \{\}$
⟨*proof*⟩

Specify the conditions for adding a new disjoint path to the set of disjoint paths.

lemma *DisjointPaths-extend*:
assumes *P-path*: $v0 \leadsto P \leadsto v1$
and *P-disjoint*: $\bigwedge xs\ v.\ [\![\ xs \in paths;\ xs \neq P;\ v \in set\ xs;\ v \in set\ P\ ]\!] \Longrightarrow v = v0 \lor v = v1$
shows *DisjointPaths G v0 v1* (*insert P paths*)
⟨*proof*⟩

lemma *DisjointPaths-reduce*:
assumes $paths' \subseteq paths$
shows *DisjointPaths G v0 v1 paths'*
⟨*proof*⟩

6.2 Second Vertices

Let us now define the set of second vertices of the paths. We are going to need this in order to find a path avoiding the old paths on its first edge.

definition *second-vertex* **where** *second-vertex* ≡ λ*xs* :: *'a Walk*. *hd* (*tl xs*)
definition *second-vertices* **where** *second-vertices* ≡ *second-vertex* ` *paths*

lemma *second-vertex-inj*: *inj-on second-vertex paths*
⟨*proof*⟩

lemma *second-vertices-card*: *card second-vertices* = *card paths*
⟨*proof*⟩

lemma *second-vertices-in-V*: *second-vertices* ⊆ *V*
⟨*proof*⟩
lemma *v0-v1-notin-second-vertices*: *v0* ∉ *second-vertices v1* ∉ *second-vertices*
⟨*proof*⟩

lemma *second-vertices-new-path*: *hd* (*tl xs*) ∉ *second-vertices* ⟹ *xs* ∉ *paths*
⟨*proof*⟩

lemma *second-vertices-first-edge*:
⟦ *xs* ∈ *paths*; *first-edge-of-walk xs* = (*v*,*w*) ⟧ ⟹ *w* ∈ *second-vertices*
⟨*proof*⟩

If we have no small separations, then the set of second vertices is not a separator and we can find a path avoiding this set.

lemma *disjoint-paths-new-path*:
assumes *no-small-separations*: ⋀*S*. *Separation G v0 v1 S* ⟹ *card S* ≥ *Suc* (*card paths*)
shows ∃ *P-new*. *v0*⇝*P-new*⇝*v1* ∧ *set P-new* ∩ *second-vertices* = {}
⟨*proof*⟩

We need the following predicate to find the first vertex on a new path that hits one of the other paths. We add the condition *x* = *v1* to cover the case *paths* = {}.

definition *hitting-paths* **where**
hitting-paths ≡ λ*x*. *x* ≠ *v0* ∧ ((∃ *xs* ∈ *paths*. *x* ∈ *set xs*) ∨ *x* = *v1*)

end — DisjointPaths

7 One More Path

Let us define a set of disjoint paths with one more path. Except for the first and last vertex, the new path must be disjoint from all other paths. The first vertex must be *v0* and the last vertex must be on some other path. In the ideal case, the last vertex will be *v1*, in which case we are already done because we have found a new disjoint path between *v0* and *v1*.

locale *DisjointPathsPlusOne* = *DisjointPaths* +
  fixes *P-new* :: *'a Walk*
  assumes *P-new*:
    $v0 \leadsto P\text{-}new \leadsto (last\ P\text{-}new)$
  and *tl-P-new*:
    $tl\ P\text{-}new \neq Nil$
    $hd\ (tl\ P\text{-}new) \notin second\text{-}vertices$
  and *last-P-new*:
    $hitting\text{-}paths\ (last\ P\text{-}new)$
    $\bigwedge v.\ v \in set\ (butlast\ P\text{-}new) \Longrightarrow \neg hitting\text{-}paths\ v$
begin

7.1 Characterizing the New Path

lemma *P-new-hd-disjoint*: $\bigwedge xs.\ xs \in paths \Longrightarrow hd\ (tl\ P\text{-}new) \neq hd\ (tl\ xs)$
  ⟨*proof*⟩

lemma *P-new-new*: $P\text{-}new \notin paths$ ⟨*proof*⟩

definition *paths-with-new* **where** $paths\text{-}with\text{-}new \equiv insert\ P\text{-}new\ paths$

lemma *card-paths-with-new*: $card\ paths\text{-}with\text{-}new = Suc\ (card\ paths)$
  ⟨*proof*⟩

lemma *paths-with-new-no-Nil*: $Nil \notin paths\text{-}with\text{-}new$
  ⟨*proof*⟩

lemma *paths-with-new-path*: $xs \in paths\text{-}with\text{-}new \Longrightarrow path\ xs$
  ⟨*proof*⟩

lemma *paths-with-new-start-in-v0*: $xs \in paths\text{-}with\text{-}new \Longrightarrow hd\ xs = v0$
  ⟨*proof*⟩

7.2 The Last Vertex of the New Path

McCuaig in [McC84] calls the last vertex of *P-new* by the name *x*. However, this name is somewhat confusing because it is so short and it will be visible in most places from now on, so let us give this vertex the more descriptive name of *new-last*.

definition *new-pre* **where** $new\text{-}pre \equiv butlast\ P\text{-}new$
definition *new-last* **where** $new\text{-}last \equiv last\ P\text{-}new$

lemma *P-new-decomp*: $P\text{-}new = new\text{-}pre\ @\ [new\text{-}last]$
  ⟨*proof*⟩

lemma *new-pre-not-Nil*: $new\text{-}pre \neq Nil$ ⟨*proof*⟩

lemma *new-pre-hitting*: $x' \in set\ new\text{-}pre \Longrightarrow \neg hitting\text{-}paths\ x'$
  ⟨*proof*⟩

lemma *P-hit*: *hitting-paths new-last*
⟨*proof*⟩

lemma *new-last-neq-v0*: *new-last* ≠ *v0* ⟨*proof*⟩

lemma *new-last-in-V*: *new-last* ∈ *V* ⟨*proof*⟩

lemma *new-last-to-v1*: ∃ *R*. *new-last* $\leadsto R \leadsto_{\textit{remove-vertex v0}}$ *v1*
⟨*proof*⟩

lemma *paths-plus-one-disjoint*:
assumes *xs* ∈ *paths-with-new ys* ∈ *paths-with-new xs* ≠ *ys v* ∈ *set xs v* ∈ *set ys*
shows *v* = *v0* ∨ *v* = *v1* ∨ *v* = *new-last*
⟨*proof*⟩

If the new path is disjoint, we are happy.

lemma *P-new-solves-if-disjoint*:
new-last = *v1* ⟹ ∃ *paths′*. *DisjointPaths G v0 v1 paths′* ∧ *card paths′* = *Suc* (*card paths*)
⟨*proof*⟩

7.3 Removing the Last Vertex

definition *H-x* **where** *H-x* ≡ *remove-vertex new-last*

lemma *H-x-Digraph*: *Digraph H-x* ⟨*proof*⟩

lemma *H-x-v0-v1-Digraph*: *new-last* ≠ *v1* ⟹ *v0-v1-Digraph H-x v0 v1* ⟨*proof*⟩

7.4 A New Path Following the Other Paths

The following lemma is one of the most complicated technical lemmas in the proof of Menger's Theorem.

Suppose we have a non-trivial path whose edges are all in the edge set of *path-with-new* and whose first edge equals the first edge of some *P* ∈ *path-with-new*. Also suppose that the path does not contain *v1* or *new-last*. Then it follows by induction that this path is an initial segment of *P*.

Note that McCuaig does not mention this statement at all in his proof because it looks so obvious.

lemma *new-path-follows-old-paths*:
assumes *xs*: *v0* $\leadsto$*xs*$\leadsto$ *w tl xs* ≠ *Nil v1* ∉ *set xs new-last* ∉ *set xs*
and *P*: *P* ∈ *paths-with-new hd* (*tl xs*) = *hd* (*tl P*)
and *edges-subset*: *edges-of-walk xs* ⊆ ⋃(*edges-of-walk* ‘ *paths-with-new*)
shows *edges-of-walk xs* ⊆ *edges-of-walk P*
⟨*proof*⟩

end — locale *DisjointPathsPlusOne*

end

8 Induction of Menger's Theorem

theory *MengerInduction* **imports** *Separations DisjointPaths* **begin**

8.1 No Small Separations

In this section we set up the general structure of the proof of Menger's Theorem. The proof is based on induction over *sep-size* (called n in McCuaig's proof), the minimum size of a separator.

locale *NoSmallSeparationsInduct* = *v0-v1-Digraph* +
  fixes *sep-size* :: *nat*
  — The size of a minimum separator.
  assumes *no-small-separations*: $\bigwedge S.$ *Separation G v0 v1 S* $\Longrightarrow$ *card S* $\geq$ *Suc sep-size*
  — The induction hypothesis.
  and *no-small-separations-hyp*: $\bigwedge G'$ :: ($'a$, $'b$) *Graph-scheme*.
    ($\bigwedge S.$ *Separation G' v0 v1 S* $\Longrightarrow$ *card S* $\geq$ *sep-size*)
    $\Longrightarrow$ *v0-v1-Digraph G' v0 v1*
    $\Longrightarrow$ $\exists$ *paths. DisjointPaths G' v0 v1 paths* $\wedge$ *card paths* = *sep-size*

Next, we want to combine this with *DisjointPathsPlusOne*.

If a minimum separator has size at least *Suc sep-size*, then it follows immediately from the induction hypothesis that we have *sep-size* many disjoint paths. We then observe that *second-vertices* of these paths is not a separator because *card second-vertices* = *sep-size*. So there exists a new path from *v0* to *v1* whose second vertex is not in *second-vertices*.

If this path is disjoint from the other paths, we have found *Suc sep-size* many disjoint paths, so assume it is not disjoint. Then there exist a vertex x on the new path that is not *v0* or *v1* such that *new-last* hits one of the other paths. Let *P-new* be the initial segment of the new path up to x. We call x, the last vertex of *P-new*, now *new-last*.

We then assume that *paths* and *P-new* have been chosen in such a way that *distance new-last v1* is minimal.

First, we define a locale that expresses that we have no small separators (with the corresponding induction hypothesis) as well as *sep-size* many internally vertex-disjoint paths (with *sep-size* $\neq$ *0* because the other case is trivial) and also one additional path that starts in *v1*, whose second vertex is not among *second-vertices* and whose last vertex is *new-last*.

We will add the assumption *new-last* $\neq$ *v1* soon.

locale *ProofStepInduct* =
  NoSmallSeparationsInduct G v0 v1 sep-size + *DisjointPathsPlusOne G v0 v1 paths P-new*
  for *G* (**structure**) **and** *v0 v1 paths P-new sep-size* +
  assumes *sep-size-not0*: *sep-size* $\neq$ *0*
    and *paths-sep-size*: *card paths* = *sep-size*

lemma (**in** *ProofStepInduct*) *hitting-paths-v1*: *hitting-paths v1*
  ⟨*proof*⟩

8.2 Choosing Paths Avoiding *new_last*

Let us now consider only the non-trivial case that *new-last* $\neq$ *v1*.

locale *ProofStepInduct-NonTrivial* = *ProofStepInduct* +
assumes *new-last-neq-v1*: *new-last* ≠ *v1*
begin

The next step is the observation that in the graph *remove-vertex new-last*, which we called *H-x*, there are also *sep-size* many internally vertex-disjoint paths, again by the induction hypothesis.

lemma *Q-exists*: ∃ *Q*. *DisjointPaths H-x v0 v1 Q* ∧ *card Q* = *sep-size*
⟨*proof*⟩

We want to choose these paths in a clever way, too. Our goal is to choose these paths such that the number of edges in ⋃ (*edges-of-walk* ‘ *Q*) ∩ (*E* − ⋃ (*edges-of-walk* ‘ *paths-with-new*)) is minimal.

definition *B* **where** *B* ≡ *E* − ⋃(*edges-of-walk* ‘ *paths-with-new*)

definition *Q-weight* **where** *Q-weight* ≡ λ*Q*. *card* (⋃(*edges-of-walk* ‘ *Q*) ∩ *B*)

definition *Q-good* **where** *Q-good* ≡ λ*Q*. *DisjointPaths H-x v0 v1 Q* ∧ *card Q* = *sep-size* ∧
(∀ *Q′*. *DisjointPaths H-x v0 v1 Q′* ∧ *card Q′* = *sep-size* ⟶ *Q-weight Q* ≤ *Q-weight Q′*)

definition *Q* **where** *Q* ≡ *SOME Q*. *Q-good Q*

It is easy to show that such a *Q* exists.

lemma *Q*: *DisjointPaths H-x v0 v1 Q card Q* = *sep-size*
and *Q-min*: ⋀*Q′*. *DisjointPaths H-x v0 v1 Q′* ∧ *card Q′* = *sep-size* ⟹ *Q-weight Q* ≤ *Q-weight Q′*
⟨*proof*⟩

sublocale *Q*: *DisjointPaths H-x v0 v1 Q* ⟨*proof*⟩

8.3 Finding a Path Avoiding *Q*

Because *Q* contains only *sep-size* many paths, we have *card Q.second-vertices* = *sep-size*. So there exists a path *P-k* among the *Suc sep-size* many paths in *paths-with-new* such that the second vertex of *P-k* is not among *Q.second-vertices*.

definition *P-k* **where**
P-k ≡ *SOME P-k*. *P-k* ∈ *paths-with-new* ∧ *hd* (*tl P-k*) ∉ *Q.second-vertices*

lemma *P-k*: *P-k* ∈ *paths-with-new hd* (*tl P-k*) ∉ *Q.second-vertices* ⟨*proof*⟩

lemma *path-P-k* [*simp*]: *path P-k* ⟨*proof*⟩
lemma *hd-P-k-v0* [*simp*]: *hd P-k* = *v0* ⟨*proof*⟩

definition *hitting-Q-or-new-last* **where**
hitting-Q-or-new-last ≡ λ*y*. *y* ≠ *v0* ∧ (*y* = *new-last* ∨ (∃ *Q-hit* ∈ *Q*. *y* ∈ *set Q-hit*))

P-k hits a vertex in *Q* or it hits *new-last* because it either ends in *v1* or in *new-last*.

lemma *P-k-hits-Q*: ∃ *y* ∈ *set P-k*. *hitting-Q-or-new-last y* ⟨*proof*⟩

end — locale *ProofStepInduct-NonTrivial*

8.4 Decomposing P_k

Having established with the previous lemma that *P-k* hits *Q* or *new-last*, let *y* be the first such vertex on *P-k*. Then we can split *P-k* at this vertex.

locale *ProofStepInduct-NonTrivial-P-k-pre* = *ProofStepInduct-NonTrivial* +
  fixes *P-k-pre y P-k-post*
  assumes *P-k-decomp*: *P-k* = *P-k-pre* @ *y* # *P-k-post*
    and *y*: *hitting-Q-or-new-last y*
    and *y-min*: $\bigwedge y'$. $y' \in$ *set P-k-pre* $\Longrightarrow$ $\neg$*hitting-Q-or-new-last* y'

We can always go from *ProofStepInduct-NonTrivial* to *ProofStepInduct-NonTrivial-P-k-pre*.

lemma (**in** *ProofStepInduct-NonTrivial*) *ProofStepInduct-NonTrivial-P-k-pre-exists*:
  shows $\exists$ *P-k-pre y P-k-post*.
    ProofStepInduct-NonTrivial-P-k-pre G v0 v1 paths P-new sep-size P-k-pre y P-k-post
⟨*proof*⟩

context *ProofStepInduct-NonTrivial-P-k-pre* **begin**
  lemma *y-neq-v0*: *y* ≠ *v0* ⟨*proof*⟩

  lemma *P-k-pre-not-Nil*: *P-k-pre* ≠ *Nil*
    ⟨*proof*⟩

  lemma *second-P-k-pre-not-in-Q*: *hd* (*tl* (*P-k-pre* @ [*y*])) ∉ *Q.second-vertices*
    ⟨*proof*⟩

  definition *H* **where** *H* ≡ *remove-vertex v0*
  sublocale *H*: *Digraph H* ⟨*proof*⟩

  lemma *y-eq-v1-implies-P-k-neq-P-new*: **assumes** *y* = *v1* **shows** *P-k* ≠ *P-new* ⟨*proof*⟩

If *y* = *v1*, then we are done.

  lemma *y-eq-v1-solves*:
    assumes *y* = *v1*
    shows ∃ *paths*. *DisjointPaths G v0 v1 paths* ∧ *card paths* = *Suc sep-size*
  ⟨*proof*⟩
end — locale *ProofStepInduct-NonTrivial-P-k-pre*

end

9 The case $y = new\_last$

theory *Y-eq-new-last* **imports** *MengerInduction* **begin**

We may assume *y* ≠ *v1* now because ⟦*ProofStepInduct-NonTrivial-P-k-pre ?G ?v0.0 ?v1.0 ?paths ?P-new ?sep-size ?P-k-pre ?y ?P-k-post*; *?y* = *?v1.0*⟧ ⟹ ∃ *paths*. *DisjointPaths ?G ?v0.0 ?v1.0 paths* ∧ *card paths* = *Suc ?sep-size* shows that *y* = *v1* already gives us *Suc sep-size* many disjoint paths.

We also assume that we have chosen the previous paths optimally in the sense that the distance from *new-last* to *v1* is minimal.

locale *ProofStepInduct-y-eq-new-last* = *ProofStepInduct-NonTrivial-P-k-pre* +
assumes *y-neq-v1*: $y \neq v1$ **and** *y-eq-new-last*: $y = new\text{-}last$
and *optimal-paths*: $\bigwedge paths'\ P\text{-}new'.$
ProofStepInduct G v0 v1 paths' P-new' sep-size
$\Longrightarrow H.distance\ (last\ P\text{-}new)\ v1 \leq H.distance\ (last\ P\text{-}new')\ v1$
begin

Let R be a shortest path from *new-last* to *v1*.

definition R **where** $R \equiv$
$SOME\ R.\ new\text{-}last \leadsto R \leadsto_H v1 \wedge (\forall R'.\ new\text{-}last \leadsto R' \leadsto_H v1 \longrightarrow length\ R \leq length\ R')$

lemma R: $new\text{-}last \leadsto R \leadsto_H v1\ \bigwedge R'.\ new\text{-}last \leadsto R' \leadsto_H v1 \Longrightarrow length\ R \leq length\ R'$ ⟨*proof*⟩

lemma *v1-in-Q*: $\exists\ Q\text{-}hit \in Q.\ v1 \in set\ Q\text{-}hit$ ⟨*proof*⟩

lemma *R-hits-Q*: $\exists\ z \in set\ R.\ Q.hitting\text{-}paths\ z$ ⟨*proof*⟩

lemma *R-decomp-exists*:
obtains *R-pre z R-post*
where $R = R\text{-}pre\ @\ z\ \#\ R\text{-}post$
and $Q.hitting\text{-}paths\ z$
and $\bigwedge z'.\ z' \in set\ R\text{-}pre \Longrightarrow \neg Q.hitting\text{-}paths\ z'$
⟨*proof*⟩

We open an anonymous context in order to hide all but the final lemma. This also gives us the decomposition of R whose existence we established above.

context fixes *R-pre z R-post*
assumes *R-decomp*: $R = R\text{-}pre\ @\ z\ \#\ R\text{-}post$
and *z*: $Q.hitting\text{-}paths\ z$
and *z-min*: $\bigwedge z'.\ z' \in set\ R\text{-}pre \Longrightarrow \neg Q.hitting\text{-}paths\ z'$
begin
private lemma *z-neq-v0*: $z \neq v0$ ⟨*proof*⟩ **lemma** *z-neq-new-last*: $z \neq new\text{-}last$ ⟨*proof*⟩ **lemma** *R-pre-neq-Nil*: $R\text{-}pre \neq Nil$ ⟨*proof*⟩ **lemma** *z-closer-than-new-last*: $H.distance\ z\ v1 < H.distance\ new\text{-}last\ v1$ ⟨*proof*⟩ **definition** *R′-walk* **where** $R'\text{-}walk \equiv P\text{-}k\text{-}pre\ @\ R\text{-}pre\ @\ [z]$

private lemma *R′-walk-not-Nil*: $R'\text{-}walk \neq Nil$ ⟨*proof*⟩ **lemma** *R′-walk-no-Q*: $[\![\ v \in set\ R'\text{-}walk;\ v \neq z\ ]\!] \Longrightarrow \neg Q.hitting\text{-}paths\ v$ ⟨*proof*⟩

The original proof goes like this: "Let z be the first vertex of R on some path in Q. Then the distance in H from z to *v1* is less than the distance from *new-last* to *v1*. This contradicts the choice of *paths* and *P-new*."

It does not say exactly why it contradicts the choice of *paths* and *P-new*. It seems we can choose Q together with *R′-walk* as our new paths plus extrapath. But this seems to be wrong because we cannot show that *R′-walk* is a path: *P-k-pre* and *R-pre* could intersect.

So we use $[\![walk\ ?xs;\ ?xs \neq [];\ hd\ ?xs = ?v;\ last\ ?xs = ?w]\!] \Longrightarrow \exists\ ys.\ ?v \leadsto ys \leadsto\ ?w \wedge set\ ys \subseteq set\ ?xs$ to transform *R′-walk* into a path R'.

private definition R' **where**
$R' \equiv SOME\ R'.\ hd\ (tl\ R'\text{-}walk) \leadsto R' \leadsto z \wedge set\ R' \subseteq set\ (tl\ R'\text{-}walk)$

private lemma R': $hd\ (tl\ R'\text{-}walk) \leadsto R' \leadsto z\ set\ R' \subseteq set\ (tl\ R'\text{-}walk)$ ⟨*proof*⟩ **lemma** *hd-R'*: $hd\ R' = hd\ (tl\ P\text{-}k)$ ⟨*proof*⟩ **lemma** *R'-no-Q*: $[\![\ v \in set\ R';\ v \neq z\ ]\!] \Longrightarrow \neg Q.hitting\text{-}paths\ v$
⟨*proof*⟩ **lemma** *v0-R'-path*: $v0 \leadsto (v0\ \#\ R') \leadsto z$ ⟨*proof*⟩ **corollary** *z-last-R'*: $z = last\ (v0\ \#\ R')$ ⟨*proof*⟩ **lemma** *z-eq-v1-solves*:
assumes $z = v1$
shows $\exists paths.\ DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = Suc\ sep\text{-}size$
⟨*proof*⟩ **lemma** *z-neq-v1-solves*:
assumes $z \neq v1$
shows $\exists paths.\ DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = Suc\ sep\text{-}size$
⟨*proof*⟩

corollary *with-optimal-paths-solves'*:
shows $\exists paths.\ DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = Suc\ sep\text{-}size$
⟨*proof*⟩
end — anonymous context

corollary *with-optimal-paths-solves*:
$\exists paths.\ DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = Suc\ sep\text{-}size$
⟨*proof*⟩

end — locale *ProofStepInduct-y-eq-new-last*
end

10 The case $y \neq new\_last$

theory *Y-neq-new-last* **imports** *MengerInduction* **begin**

Let us now consider the case that $y \neq v1 \wedge y \neq new\text{-}last$. Our goal is to show that this is inconsistent: The following locale will be unsatisfiable, proving that $y = v1 \vee y = new\text{-}last$ holds.

locale *ProofStepInduct-y-neq-new-last* = *ProofStepInduct-NonTrivial-P-k-pre* +
assumes *y-neq-v1*: $y \neq v1$ **and** *y-neq-new-last*: $y \neq new\text{-}last$
begin

lemma *Q-hit-exists*: **obtains** *Q-hit Q-hit-pre Q-hit-post* **where**
$Q\text{-}hit \in Q\ y \in set\ Q\text{-}hit\ Q\text{-}hit = Q\text{-}hit\text{-}pre\ @\ y\ \#\ Q\text{-}hit\text{-}post$
⟨*proof*⟩

We open an anonymous context because we do not want to export any lemmas except the final lemma proving the contradiction. This is also an easy way to get the decomposition of *Q-hit*, whose existence we have established above.

context
fixes *Q-hit Q-hit-pre Q-hit-post*
assumes *Q-hit*: $Q\text{-}hit \in Q\ y \in set\ Q\text{-}hit$
and *Q-hit-decomp*: $Q\text{-}hit = Q\text{-}hit\text{-}pre\ @\ y\ \#\ Q\text{-}hit\text{-}post$
begin
private lemma *Q-hit-v0-v1*: $v0 \leadsto Q\text{-}hit \leadsto_{H\text{-}x} v1$ ⟨*proof*⟩ **lemma** *Q-hit-vertices*: $set\ Q\text{-}hit \subseteq V - \{new\text{-}last\}$
⟨*proof*⟩ **lemma** *Q-hit-pre-not-Nil*: $Q\text{-}hit\text{-}pre \neq Nil$

⟨*proof*⟩ **lemma** *tl-Q-hit-pre*: *tl* (*Q-hit-pre* @ [*y*]) ≠ *Nil* ⟨*proof*⟩ **lemma** *Q-hit-pre-edges*: *edges-of-walk* (*Q-hit-pre* @ [*y*]) ∩ *B* ≠ {} ⟨*proof*⟩ **lemma** *P-k-pre-edges*: *edges-of-walk* (*P-k-pre* @ [*y*]) ∩ *B* = {} ⟨*proof*⟩ **definition** *Q-hit′* **where** *Q-hit′* ≡ *P-k-pre* @ *y* # *Q-hit-post*

private lemma *Q-hit′-v0-v1*: *v0* ⇝*Q-hit′*⇝ *v1* ⟨*proof*⟩ **lemma** *Q-hit′-v0-v1-H-x*: *v0* ⇝*Q-hit′*⇝$_{H\text{-}x}$ *v1* ⟨*proof*⟩ **definition** *Q′* **where** *Q′* ≡ *insert Q-hit′* (*Q* − {*Q-hit*})

private lemma *Q-hit-edges-disjoint*:
⋃(*edges-of-walk* ‘ (*Q* − {*Q-hit*})) ∩ *edges-of-walk Q-hit* = {}
⟨*proof*⟩ **lemma** *Q-hit′-notin-Q-minus-Q-hit*: *Q-hit′* ∉ *Q* − {*Q-hit*} ⟨*proof*⟩ **lemma** *Q-weight-smaller*: *Q-weight Q′* < *Q-weight Q* ⟨*proof*⟩ **lemma** *DisjointPaths-Q′*: *DisjointPaths H-x v0 v1 Q′* ⟨*proof*⟩ **lemma** *card-Q′*: *card Q′* = *sep-size* ⟨*proof*⟩

lemma *contradiction′*: *False* ⟨*proof*⟩
end — anonymous context

corollary *contradiction*: *False* ⟨*proof*⟩

end — locale *ProofStepInduct-y-neq-new-last*
end

11 Menger's Theorem

theory *Menger* **imports** *Y-eq-new-last Y-neq-new-last* **begin**

In this section, we combine the cases and finally prove Menger's Theorem.

locale *ProofStepInductOptimalPaths* = *ProofStepInduct* +
assumes *optimal-paths*:
⋀*paths′ P-new′*. *ProofStepInduct G v0 v1 paths′ P-new′ sep-size*
⟹ *Digraph.distance* (*remove-vertex v0*) (*last P-new*) *v1*
≤ *Digraph.distance* (*remove-vertex v0*) (*last P-new′*) *v1*
begin

lemma *one-more-paths-exists-trivial*:
new-last = *v1* ⟹ ∃ *paths*. *DisjointPaths G v0 v1 paths* ∧ *card paths* = *Suc sep-size*
⟨*proof*⟩

lemma *one-more-paths-exists-nontrivial*:
assumes *new-last* ≠ *v1*
shows ∃ *paths*. *DisjointPaths G v0 v1 paths* ∧ *card paths* = *Suc sep-size*
⟨*proof*⟩

corollary *one-more-paths-exists*:
shows ∃ *paths*. *DisjointPaths G v0 v1 paths* ∧ *card paths* = *Suc sep-size*
⟨*proof*⟩

end

lemma (**in** *ProofStepInduct*) *one-more-paths-exists*:
∃ *paths*. *DisjointPaths G v0 v1 paths* ∧ *card paths* = *Suc sep-size*
⟨*proof*⟩

11.1 Menger's Theorem

theorem (**in** *v0-v1-Digraph*) *menger*:
  assumes $\bigwedge S.\ Separation\ G\ v0\ v1\ S \Longrightarrow card\ S \geq n$
  shows $\exists paths.\ DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = n$
⟨*proof*⟩

The previous theorem was the difficult direction of Menger's Theorem. Let us now prove the other direction: If we have n disjoint paths, than every separator must contain at least n vertices. This direction is rather trivial because every separator needs to separate at least the n paths, so we do not need induction or an elaborate setup to prove this.

theorem (**in** *v0-v1-Digraph*) *menger-trivial*:
  assumes $DisjointPaths\ G\ v0\ v1\ paths\ card\ paths = n$
  shows $\bigwedge S.\ Separation\ G\ v0\ v1\ S \Longrightarrow card\ S \geq n$
⟨*proof*⟩

11.2 Self-contained Statement of the Main Theorem

Let us state both directions of Menger's Theorem again in a more self-contained way in the *Digraph* locale. Stating the theorems in a self-contained way helps avoiding mistakes due to wrong definitions hidden in one of the numerous locales we used and also significantly reduces the work needed to review this formalization.

With the statements below, all you need to do in order to verify that this formalization actually expresses Menger's Theorem (and not something else), is to look into the assumptions and definitions of the *Digraph* locale.

theorem (**in** *Digraph*) *menger*:
  fixes $v0\ v1 :: 'a$ **and** $n :: nat$
  assumes *v0-V*: $v0 \in V$
    and *v1-V*: $v1 \in V$
    and *v0-nonadj-v1*: $\neg v0 \rightarrow v1$
    and *v0-neq-v1*: $v0 \neq v1$
    and *no-small-separators*: $\bigwedge S.$
      $[\![\ S \subseteq V;\ v0 \notin S;\ v1 \notin S;\ \bigwedge xs.\ v0 \leadsto xs \leadsto v1 \Longrightarrow set\ xs \cap S \neq \{\}\ ]\!] \Longrightarrow card\ S \geq n$
  shows $\exists paths.\ card\ paths = n \wedge (\forall xs \in paths.$
    $v0 \leadsto xs \leadsto v1 \wedge (\forall ys \in paths - \{xs\}.\ (\forall v \in set\ xs \cap set\ ys.\ v = v0 \vee v = v1)))$
⟨*proof*⟩

theorem (**in** *Digraph*) *menger-trivial*:
  fixes $v0\ v1 :: 'a$ **and** $n :: nat$
  assumes *v0-V*: $v0 \in V$
    and *v1-V*: $v1 \in V$
    and *v0-nonadj-v1*: $\neg v0 \rightarrow v1$
    and *v0-neq-v1*: $v0 \neq v1$
    and *n-paths*: $card\ paths = n$
    and *paths-disjoint*: $\forall xs \in paths.$
      $v0 \leadsto xs \leadsto v1 \wedge (\forall ys \in paths - \{xs\}.\ (\forall v \in set\ xs \cap set\ ys.\ v = v0 \vee v = v1))$
  shows $\bigwedge S.\ [\![\ S \subseteq V;\ v0 \notin S;\ v1 \notin S;\ \bigwedge xs.\ v0 \leadsto xs \leadsto v1 \Longrightarrow set\ xs \cap S \neq \{\}\ ]\!] \Longrightarrow card\ S \geq n$
⟨*proof*⟩

end

References

[Loc16] Andreas Lochbihler. A formal proof of the max-flow min-cut theorem for countable networks. *Archive of Formal Proofs*, May 2016. http://isa-afp.org/entries/MFMC_Countable.shtml, Formal proof development.

[McC84] William McCuaig. A simple proof of Menger's theorem. *Journal of Graph Theory*, 8(3):427–429, 1984. `doi:10.1002/jgt.3190080311`.

[Men27] Karl Menger. Zur allgemeinen Kurventheorie. *Fundamenta Mathematicae*, 10(1):96–115, 1927. URL: http://eudml.org/doc/211191.