

Menger's Theorem

Christoph Dittmann
isabelle@christoph-d.de

May 26, 2024

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 <i>new_last</i>	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 $\text{card } (A \cap X) = \text{card } (B \cap X) + \text{card } (C \cap X)$
<proof>

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

lemma *set-butlast*: $\llbracket x \in \text{set } xs; x \neq \text{last } xs \rrbracket \implies x \in \text{set } (\text{butlast } xs)$
<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 $P :: 'a \Rightarrow \text{bool}$ **and** $\text{weight} :: 'a \Rightarrow \text{nat}$
assumes $\exists x. P x$
obtains x **where** $P x \wedge \forall y. P y \implies \text{weight } x \leq \text{weight } y$
<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 $'a \text{ Edge} = 'a \times 'a$
type-synonym $'a \text{ Walk} = 'a \text{ list}$

record $'a \text{ Graph} =$

verts :: $'a \text{ set } (V_1)$
arcs :: $'a \text{ Edge set } (E_1)$

abbreviation *is-arc* :: $('a, 'b) \text{ Graph-scheme} \Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ (**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  $\Rightarrow$  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:  $\llbracket$  walk  $xs$ ; walk  $ys$ ;  $xs = Nil \vee ys = Nil \vee$  last  $xs \rightarrow hd$   $ys$   $\rrbracket \Longrightarrow$  walk ( $xs @ ys$ )
  <proof>
lemma walk-tl: walk  $xs \Longrightarrow$  walk (tl  $xs$ ) <proof>
lemma walk-drop: walk  $xs \Longrightarrow$  walk (drop  $n$   $xs$ ) <proof>
lemma walk-take: walk  $xs \Longrightarrow$  walk (take  $n$   $xs$ )
  <proof>
lemma walk-decomp: assumes walk ( $xs @ ys$ ) shows walk  $xs$  walk  $ys$ 
  <proof>
lemma walk-in- $V$ : walk  $xs \Longrightarrow$  set  $xs \subseteq V$  <proof>
lemma walk-first-edge: walk ( $v \# w \# xs$ )  $\Longrightarrow$   $v \rightarrow w$  <proof>
lemma walk-first-edge':  $\llbracket$  walk ( $v \# xs$ );  $xs \neq Nil$   $\rrbracket \Longrightarrow$   $v \rightarrow hd$   $xs$ 
  <proof>
lemma walk-middle-edge: walk ( $xs @ v \# w \# ys$ )  $\Longrightarrow$   $v \rightarrow w$ 
  <proof>
lemma walk-last-edge:  $\llbracket$  walk ( $xs @ ys$ );  $xs \neq Nil$ ;  $ys \neq Nil$   $\rrbracket \Longrightarrow$  last  $xs \rightarrow hd$   $ys$ 
  <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  $\Rightarrow$  bool where path  $xs \equiv$  walk  $xs \wedge$  distinct  $xs$ 

```

```

lemma path-singleton [simp]:  $v \in V \Longrightarrow$  path [ $v$ ] <proof>
lemma path-2 [simp]:  $\llbracket$   $v \rightarrow w$ ;  $v \neq w$   $\rrbracket \Longrightarrow$  path [ $v, w$ ] <proof>
lemma path-cons:  $\llbracket$  path  $xs$ ;  $xs \neq Nil$ ;  $v \rightarrow hd$   $xs$ ;  $v \notin$  set  $xs$   $\rrbracket \Longrightarrow$  path ( $v \# xs$ )

```

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

4.3 The Set of All Paths

definition all-paths where $\text{all-paths} \equiv \{ xs \mid xs. \text{path } xs \}$

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* $\langle \text{proof} \rangle$

end — context Digraph

We introduce shorthand notation for a path connecting two vertices.

definition path-from-to $:: ('a, 'b) \text{Graph-scheme} \Rightarrow 'a \Rightarrow 'a \text{Walk} \Rightarrow 'a \Rightarrow \text{bool}$
 $(- \rightsquigarrow \rightsquigarrow_1 - [71, 71, 71] 70)$ **where**
 $\text{path-from-to } G \ v \ xs \ w \equiv \text{Digraph.path } G \ xs \wedge xs \neq \text{Nil} \wedge \text{hd } xs = v \wedge \text{last } xs = w$

context Digraph begin

lemma path-from-toI [*intro*]: $\llbracket \text{path } xs; xs \neq \text{Nil}; \text{hd } xs = v; \text{last } xs = w \rrbracket \implies v \rightsquigarrow xs \rightsquigarrow w$
and path-from-toE [*dest*]: $v \rightsquigarrow xs \rightsquigarrow w \implies \text{path } xs \wedge xs \neq \text{Nil} \wedge \text{hd } xs = v \wedge \text{last } xs = w$
 $\langle \text{proof} \rangle$

lemma path-from-to-ends: $v \rightsquigarrow (xs @ w \# ys) \rightsquigarrow w \implies ys = \text{Nil}$
 $\langle \text{proof} \rangle$

lemma path-from-to-combine:

assumes $v \rightsquigarrow (xs @ x \# xs') \rightsquigarrow w \ v' \rightsquigarrow (ys @ x \# ys') \rightsquigarrow w'$ $\text{set } xs \cap \text{set } ys' = \{ \}$
shows $v \rightsquigarrow (xs @ x \# ys') \rightsquigarrow w'$
 $\langle \text{proof} \rangle$

lemma path-from-to-first: $v \rightsquigarrow xs \rightsquigarrow w \implies v \notin \text{set } (\text{tl } xs)$
 $\langle \text{proof} \rangle$

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

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

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

Every walk contains a path connecting the same vertices.

lemma *walk-to-path*:
assumes $\text{walk } xs \text{ } xs \neq \text{Nil } \text{hd } xs = v \text{ } \text{last } xs = w$
shows $\exists ys. v \rightsquigarrow ys \rightsquigarrow w \wedge \text{set } ys \subseteq \text{set } xs$
 ⟨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-of-walk* :: 'a Walk \Rightarrow 'a Edge set **where**
 $\text{edges-of-walk } xs = \{ (v,w) \mid v \text{ } w \text{ } xs\text{-pre } xs\text{-post}. xs = xs\text{-pre } @ v \# w \# xs\text{-post} \}$

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

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

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

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

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

lemma *edges-of-walk-edge*: $\llbracket \text{walk } xs; (v,w) \in \text{edges-of-walk } xs \rrbracket \implies v \rightarrow w$
 ⟨proof⟩

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

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

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

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

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

<proof>

lemma *walk-edges-decomp'*:

edges-of-walk ($xs @ v \# w \# xs'$) = *edges-of-walk* ($xs @ [v]$) \cup $\{(v,w)\}$ \cup *edges-of-walk* ($w \# xs'$)

<proof>

lemma *walk-edges-vertices*: **assumes** $(v, w) \in$ *edges-of-walk* xs **shows** $v \in$ *set* xs $w \in$ *set* xs

<proof>

lemma *walk-edges-subset*:

assumes *edges-subsets*: *edges-of-walk* $xs \subseteq$ *edges-of-walk* ys

and *non-trivial*: tl $xs \neq Nil$

shows *set* $xs \subseteq$ *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 $(v,w) \in$ *edges-of-walk* xs

shows \exists *xs-pre* *xs-post*. $xs = xs\text{-pre} @ v \# w \# xs\text{-post}$

$\wedge (v,w) \notin$ *edges-of-walk* ($xs\text{-pre} @ [v]$)

$\wedge (v,w) \notin$ *edges-of-walk* ($w \# xs\text{-post}$)

<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 $xs \neq Nil \implies$ *first-edge-of-walk* $xs \in$ *edges-of-walk* xs

<proof>

lemma *first-edge-hd-tl*: $\llbracket v \rightsquigarrow xs \rightsquigarrow w; tl$ $xs \neq Nil \rrbracket \implies$ *first-edge-of-walk* $xs = (v, hd$ (tl xs))

<proof>

lemma *first-edge-first*:

assumes $v \rightsquigarrow xs \rightsquigarrow w$ $(v,w') \in$ *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 ⇒ 'a ⇒ nat **where**
distance v w ≡ Min { length xs | xs. v ↪ xs ↪ 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 | xs. v ↪ xs ↪ w } ⟨proof⟩

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 ↪ xs ↪ w ⇒ distance v w ≤ length xs
 ⟨proof⟩

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 ↪ xs ↪ w
and xs-min: ∀ xs'. v ↪ xs' ↪ w ⇒ length xs ≤ length xs'
shows distance v w = length xs
 ⟨proof⟩

4.7 Subgraphs

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

definition *remove-vertex* :: 'a ⇒ ('a, 'b) Graph-scheme **where**
remove-vertex x ≡ G(| verts := V - {x}, arcs := Restr E (V - {x}) |)

lemma *remove-vertex-V*: V_remove-vertex x = V - {x} ⟨proof⟩

lemma *remove-vertex-V'*: V_remove-vertex x ⊆ V ⟨proof⟩

lemma *remove-vertex-E*: E_remove-vertex x = Restr E (V - {x}) ⟨proof⟩

lemma *remove-vertex-E'*: v →_remove-vertex x w ⇒ v → w ⟨proof⟩

lemma *remove-vertex-E''*: [v → w; v ≠ x; w ≠ x] ⇒ v →_remove-vertex x w
 ⟨proof⟩

Of course, this is still a digraph.

lemma *remove-vertex-Digraph*: Digraph (remove-vertex v) ⟨proof⟩

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 ∉ set xs
shows Digraph.walk (remove-vertex x) xs
 ⟨proof⟩

The same holds for paths.

lemma *remove-vertex-path-from-to*:
 $\llbracket v \rightsquigarrow xs \rightsquigarrow w; x \in V; x \notin \text{set } xs \rrbracket \implies v \rightsquigarrow xs \rightsquigarrow_{\text{remove-vertex } x} w$
 <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*: $v \rightsquigarrow xs \rightsquigarrow_{\text{remove-vertex } x} w \implies v \rightsquigarrow xs \rightsquigarrow w$
 <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 undirected 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 $v0\text{-notin-}S: v0 \notin S$
and $v1\text{-notin-}S: v1 \notin S$
and $S\text{-separates}: \bigwedge xs. v0 \rightsquigarrow xs \rightsquigarrow v1 \implies \text{set } xs \cap S \neq \{\}$

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: $\bigwedge S. \text{Separation } G v0 v1 S \implies \text{card } S \geq \text{Suc } n$*
and *Separation (remove-vertex v) v0 v1 S*
shows $\text{card } S \geq n$
<proof>

lemma (in *v0-v1-Digraph*) *path-exists-if-no-separation:*

assumes $S \subseteq V$ $v0 \notin S$ $v1 \notin S$ $\neg \text{Separation } G v0 v1 S$
shows $\exists xs. v0 \rightsquigarrow xs \rightsquigarrow v1 \wedge \text{set } xs \cap S = \{\}$
<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*:

$\bigwedge xs. xs \in paths \implies v0 \rightsquigarrow xs \rightsquigarrow v1$

and *paths-disjoint*: $\bigwedge xs\ ys\ v.$

$\llbracket xs \in paths; ys \in paths; xs \neq ys; v \in set\ xs; v \in set\ ys \rrbracket \implies v = v0 \vee v = v1$

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-paths* *<proof>*

lemma *finite-paths*: *finite* *paths*

<proof>

lemma *paths-edge-finite*: *finite* (\bigcup (*edges-of-walk* ‘*paths*’)) *<proof>*

lemma *paths-tl-notnil*: *xs* \in *paths* \implies *tl* *xs* \neq *Nil*

<proof>

lemma *paths-second-in-V*: *xs* \in *paths* \implies *hd* (*tl* *xs*) \in *V*

<proof>

lemma *paths-second-not-v0*: *xs* \in *paths* \implies *hd* (*tl* *xs*) \neq *v0*

<proof>

lemma *paths-second-not-v1*: *xs* \in *paths* \implies *hd* (*tl* *xs*) \neq *v1*

<proof>

lemma *paths-second-disjoint*: $\llbracket xs \in paths; ys \in paths; xs \neq ys \rrbracket \implies hd\ (tl\ xs) \neq hd\ (tl\ ys)$

<proof>

lemma *paths-edge-disjoint*:

assumes *xs* \in *paths* *ys* \in *paths* *xs* \neq *ys*

shows *edges-of-walk* *xs* \cap *edges-of-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* \rightsquigarrow *P* \rightsquigarrow *v1*

and *P-disjoint*: $\bigwedge xs\ v. \llbracket xs \in paths; xs \neq P; v \in set\ xs; v \in set\ P \rrbracket \implies v = v0 \vee v = v1$

shows *DisjointPaths* *G* *v0* *v1* (*insert* *P* *paths*)

<proof>

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

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** $second-vertex \equiv \lambda xs :: 'a\ Walk.\ hd\ (tl\ xs)$

definition *second-vertices* **where** $second-vertices \equiv second-vertex\ 'paths$

lemma *second-vertex-inj*: $inj\ on\ second-vertex\ paths$
 $\langle proof \rangle$

lemma *second-vertices-card*: $card\ second-vertices = card\ paths$
 $\langle proof \rangle$

lemma *second-vertices-in-V*: $second-vertices \subseteq V$
 $\langle proof \rangle$

lemma *v0-v1-notin-second-vertices*: $v0 \notin second-vertices\ v1 \notin second-vertices$
 $\langle proof \rangle$

lemma *second-vertices-new-path*: $hd\ (tl\ xs) \notin second-vertices \implies xs \notin paths$
 $\langle proof \rangle$

lemma *second-vertices-first-edge*:
 $\llbracket xs \in paths; first-edge-of-walk\ xs = (v,w) \rrbracket \implies w \in second-vertices$
 $\langle proof \rangle$

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*: $\bigwedge S.\ Separation\ G\ v0\ v1\ S \implies card\ S \geq Suc\ (card\ paths)$
shows $\exists P\ new.\ v0 \rightsquigarrow P\ new \rightsquigarrow v1 \wedge set\ P\ new \cap second-vertices = \{\}$
 $\langle proof \rangle$

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**
 $hitting-paths \equiv \lambda x.\ x \neq v0 \wedge ((\exists xs \in paths.\ x \in set\ xs) \vee x = v1)$

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  $\rightsquigarrow$ P-new $\rightsquigarrow$  (last P-new)
  and tl-P-new:
    tl P-new  $\neq$  Nil
    hd (tl P-new)  $\notin$  second-vertices
  and last-P-new:
    hitting-paths (last P-new)
     $\bigwedge v. v \in \text{set } (\text{butlast } P\text{-new}) \implies \neg \text{hitting-paths } v$ 
begin

```

7.1 Characterizing the New Path

lemma P-new-hd-disjoint: $\bigwedge xs. xs \in \text{paths} \implies \text{hd } (\text{tl } P\text{-new}) \neq \text{hd } (\text{tl } xs)$
 <proof>

lemma P-new-new: $P\text{-new} \notin \text{paths}$ <proof>

definition paths-with-new **where** paths-with-new \equiv insert P-new paths

lemma card-paths-with-new: $\text{card } \text{paths-with-new} = \text{Suc } (\text{card } \text{paths})$
 <proof>

lemma paths-with-new-no-Nil: $\text{Nil} \notin \text{paths-with-new}$
 <proof>

lemma paths-with-new-path: $xs \in \text{paths-with-new} \implies \text{path } xs$
 <proof>

lemma paths-with-new-start-in-v0: $xs \in \text{paths-with-new} \implies \text{hd } xs = v0$
 <proof>

7.2 The Last Vertex of the New Path

McCuaig in [McC84] calls the last vertex of $P\text{-new}$ 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-pre \equiv butlast P-new

definition new-last **where** new-last \equiv last P-new

lemma P-new-decomp: $P\text{-new} = \text{new-pre} @ [\text{new-last}]$
 <proof>

lemma new-pre-not-Nil: $\text{new-pre} \neq \text{Nil}$ <proof>

lemma new-pre-hitting: $x' \in \text{set } \text{new-pre} \implies \neg \text{hitting-paths } x'$
 <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 \rightsquigarrow R \rightsquigarrow 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 \implies ∃ paths'. DisjointPaths G v0 v1 paths' ∧ card paths' = Suc (card paths)

⟨proof⟩

7.3 Removing the Last Vertex

definition *H-x where H-x \equiv remove-vertex new-last*

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

lemma *H-x-v0-v1-Digraph: new-last ≠ v1 \implies 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 $P \in \text{path-with-new}$. 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 \rightsquigarrow xs \rightsquigarrow w tl xs ≠ Nil v1 \notin set xs new-last \notin set xs*

and *P: P ∈ paths-with-new hd (tl xs) = hd (tl P)*

and *edges-subset: edges-of-walk xs \subseteq \bigcup (edges-of-walk ' paths-with-new)*

shows *edges-of-walk xs \subseteq 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*: $\bigwedge S. \text{Separation } G \ v0 \ v1 \ S \implies \text{card } S \geq \text{Suc } \text{sep-size}$
 — The induction hypothesis.
and *no-small-separations-hyp*: $\bigwedge G' :: ('a, 'b) \text{Graph-scheme.}$
 ($\bigwedge S. \text{Separation } G' \ v0 \ v1 \ S \implies \text{card } S \geq \text{sep-size}$)
 $\implies \text{v0-v1-Digraph } G' \ v0 \ v1$
 $\implies \exists \text{paths. DisjointPaths } G' \ v0 \ v1 \ \text{paths} \wedge \text{card } \text{paths} = \text{sep-size}$

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

If a minimum separator has size at least $\text{Suc } \text{sep-size}$, 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 $\text{card } \text{second-vertices} = \text{sep-size}$. 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 $\text{Suc } \text{sep-size}$ 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 $\text{sep-size} \neq 0$ 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 $\text{new-last} \neq v1$ soon.

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

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 $\text{new-last} \neq v1$.

locale *ProofStepInduct-NonTrivial* = *ProofStepInduct* +
assumes *new-last-neq-v1*: *new-last* \neq *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*: $\exists Q. \text{DisjointPaths } H-x \ v0 \ v1 \ Q \wedge \text{card } Q = \text{sep-size}$
 $\langle \text{proof} \rangle$

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 $\bigcup (\text{edges-of-walk } 'Q) \cap (E - \bigcup (\text{edges-of-walk } ' \text{paths-with-new}))$ is minimal.

definition *B* **where** $B \equiv E - \bigcup (\text{edges-of-walk } ' \text{paths-with-new})$

definition *Q-weight* **where** $Q\text{-weight} \equiv \lambda Q. \text{card } (\bigcup (\text{edges-of-walk } ' Q) \cap B)$

definition *Q-good* **where** $Q\text{-good} \equiv \lambda Q. \text{DisjointPaths } H-x \ v0 \ v1 \ Q \wedge \text{card } Q = \text{sep-size} \wedge$
 $(\forall Q'. \text{DisjointPaths } H-x \ v0 \ v1 \ Q' \wedge \text{card } Q' = \text{sep-size} \longrightarrow Q\text{-weight } Q \leq Q\text{-weight } Q')$

definition *Q* **where** $Q \equiv \text{SOME } Q. Q\text{-good } Q$

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

lemma *Q*: $\text{DisjointPaths } H-x \ v0 \ v1 \ Q \wedge \text{card } Q = \text{sep-size}$
and *Q-min*: $\bigwedge Q'. \text{DisjointPaths } H-x \ v0 \ v1 \ Q' \wedge \text{card } Q' = \text{sep-size} \implies Q\text{-weight } Q \leq Q\text{-weight } Q'$
 $\langle \text{proof} \rangle$

sublocale *Q*: $\text{DisjointPaths } H-x \ v0 \ v1 \ Q \langle \text{proof} \rangle$

8.3 Finding a Path Avoiding *Q*

Because *Q* contains only *sep-size* many paths, we have $\text{card } Q.\text{second-vertices} = \text{sep-size}$. 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**

$P-k \equiv \text{SOME } P-k. P-k \in \text{paths-with-new} \wedge \text{hd } (tl \ P-k) \notin Q.\text{second-vertices}$

lemma *P-k*: $P-k \in \text{paths-with-new} \wedge \text{hd } (tl \ P-k) \notin Q.\text{second-vertices} \langle \text{proof} \rangle$

lemma *path-P-k* [*simp*]: $\text{path } P-k \langle \text{proof} \rangle$

lemma *hd-P-k-v0* [*simp*]: $\text{hd } P-k = v0 \langle \text{proof} \rangle$

definition *hitting-Q-or-new-last* **where**

$\text{hitting-Q-or-new-last} \equiv \lambda y. y \neq v0 \wedge (y = \text{new-last} \vee (\exists Q\text{-hit} \in Q. y \in \text{set } Q\text{-hit}))$

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*: $\exists y \in \text{set } P-k. \text{hitting-Q-or-new-last } y \langle \text{proof} \rangle$

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:  $\bigwedge y'. y' \in \text{set } P-k\text{-pre} \implies \neg \text{hitting-Q-or-new-last } y'$ 

```

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

```

lemma (in ProofStepInduct-NonTrivial) ProofStepInduct-NonTrivial-P-k-pre-exists:
  shows  $\exists P-k\text{-pre } y P-k\text{-post}$ .

```

```

  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:  $y \neq v0$  <proof>

```

```

  lemma P-k-pre-not-Nil:  $P-k\text{-pre} \neq \text{Nil}$ 
  <proof>

```

```

  lemma second-P-k-pre-not-in-Q:  $\text{hd } (\text{tl } (P-k\text{-pre } @ [y])) \notin Q.\text{second-vertices}$ 
  <proof>

```

```

  definition H where  $H \equiv \text{remove-vertex } v0$ 
  sublocale H: Digraph H <proof>

```

```

  lemma y-eq-v1-implies-P-k-neq-P-new: assumes  $y = v1$  shows  $P-k \neq P\text{-new}$  <proof>

```

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

```

  lemma y-eq-v1-solves:
    assumes  $y = v1$ 
    shows  $\exists \text{paths. DisjointPaths } G v0 v1 \text{ paths} \wedge \text{card paths} = \text{Suc } \text{sep-size}$ 
  <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 $y \neq v1$ now because $\llbracket \text{ProofStepInduct-NonTrivial-P-k-pre } ?G \text{ } ?v0.0 \text{ } ?v1.0 \text{ } ?\text{paths } ?P\text{-new } ?\text{sep-size } ?P\text{-pre } ?y \text{ } ?P\text{-post}; ?y = ?v1.0 \rrbracket \implies \exists \text{paths. DisjointPaths } ?G \text{ } ?v0.0 \text{ } ?v1.0 \text{ paths} \wedge \text{card paths} = \text{Suc } ?\text{sep-size}$ 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*: $y = new-last$
and *optimal-paths*: $\bigwedge paths' P-new'$
ProofStepInduct G v0 v1 paths' P-new' sep-size
 $\implies H.distance (last P-new) v1 \leq H.distance (last P-new') v1$
begin

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

definition R **where** $R \equiv$

SOME R. new-last $\rightsquigarrow_{R \rightsquigarrow_H} v1 \wedge (\forall R'. new-last \rightsquigarrow_{R' \rightsquigarrow_H} v1 \implies length R \leq length R')$

lemma R : *new-last $\rightsquigarrow_{R \rightsquigarrow_H} v1 \wedge R'. new-last \rightsquigarrow_{R' \rightsquigarrow_H} v1 \implies length R \leq length R'$* *<proof>*

lemma *v1-in-Q*: $\exists Q-hit \in Q. v1 \in set Q-hit$ *<proof>*

lemma *R-hits-Q*: $\exists z \in set R. Q.hitting-paths z$ *<proof>*

lemma *R-decomp-exists*:

obtains $R-pre z R-post$

where $R = R-pre @ z \# R-post$

and $Q.hitting-paths z$

and $\bigwedge z'. z' \in set R-pre \implies \neg Q.hitting-paths z'$

<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*: $R = R-pre @ z \# R-post$

and $z: Q.hitting-paths z$

and $z-min$: $\bigwedge z'. z' \in set R-pre \implies \neg Q.hitting-paths z'$

begin

private lemma *z-neq-v0*: $z \neq v0$ *<proof>* **lemma** *z-neq-new-last*: $z \neq new-last$ *<proof>* **lemma** *R-pre-neq-Nil*: $R-pre \neq Nil$ *<proof>* **lemma** *z-closer-than-new-last*: $H.distance z v1 < H.distance new-last v1$ *<proof>* **definition** $R'-walk$ **where** $R'-walk \equiv P-k-pre @ R-pre @ [z]$

private lemma *R'-walk-not-Nil*: $R'-walk \neq Nil$ *<proof>* **lemma** *R'-walk-no-Q*: $\llbracket v \in set R'-walk; v \neq z \rrbracket \implies \neg Q.hitting-paths v$ *<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 $\llbracket walk ?xs; ?xs \neq []; hd ?xs = ?v; last ?xs = ?w \rrbracket \implies \exists ys. ?v \rightsquigarrow ys \rightsquigarrow ?w \wedge set ys \subseteq set ?xs$ to transform $R'-walk$ into a path R' .

private definition R' **where**

$R' \equiv SOME R'. hd (tl R'-walk) \rightsquigarrow_{R' \rightsquigarrow} z \wedge set R' \subseteq set (tl R'-walk)$

```

private lemma  $R'$ :  $hd (tl R'\text{-walk}) \rightsquigarrow R' \rightsquigarrow z$   $set R' \subseteq set (tl R'\text{-walk})$   $\langle proof \rangle$  lemma  $hd\text{-}R'$ :  $hd$ 
 $R' = hd (tl P\text{-}k)$   $\langle proof \rangle$  lemma  $R'\text{-no-}Q$ :  $\llbracket v \in set R'; v \neq z \rrbracket \implies \neg Q.\text{hitting-paths } v$ 
 $\langle proof \rangle$  lemma  $v0\text{-}R'\text{-path}$ :  $v0 \rightsquigarrow (v0 \# R') \rightsquigarrow z$   $\langle proof \rangle$  corollary  $z\text{-last-}R'$ :  $z = last (v0 \# R')$ 
 $\langle proof \rangle$  lemma  $z\text{-eq-}v1\text{-solves}$ :
  assumes  $z = v1$ 
  shows  $\exists paths. DisjointPaths G v0 v1 paths \wedge card paths = Suc sep\text{-}size$ 
 $\langle proof \rangle$  lemma  $z\text{-neq-}v1\text{-solves}$ :
  assumes  $z \neq v1$ 
  shows  $\exists paths. DisjointPaths G v0 v1 paths \wedge card paths = Suc sep\text{-}size$ 
 $\langle proof \rangle$ 

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

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

end — locale  $ProofStepInduct\text{-}y\text{-eq-}new\text{-}last$ 
end

```

10 The case $y \neq new_last$

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

```

```

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

```

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 $Q\text{-hit}$, whose existence we have established above.

```

context
  fixes  $Q\text{-hit}$   $Q\text{-hit-pre}$   $Q\text{-hit-post}$ 
  assumes  $Q\text{-hit}$ :  $Q\text{-hit} \in Q$   $y \in set Q\text{-hit}$ 
  and  $Q\text{-hit-decomp}$ :  $Q\text{-hit} = Q\text{-hit-pre} @ y \# Q\text{-hit-post}$ 
begin
  private lemma  $Q\text{-hit-v0-v1}$ :  $v0 \rightsquigarrow Q\text{-hit} \rightsquigarrow_{H\text{-}x} v1$   $\langle proof \rangle$  lemma  $Q\text{-hit-vertices}$ :  $set Q\text{-hit} \subseteq V$ 
  —  $\{new\text{-}last\}$ 
   $\langle proof \rangle$  lemma  $Q\text{-hit-pre-not-}Nil$ :  $Q\text{-hit-pre} \neq Nil$ 

```

$\langle \text{proof} \rangle$ **lemma** *tl-Q-hit-pre*: $tl (Q\text{-hit-pre } @ [y]) \neq Nil$ $\langle \text{proof} \rangle$ **lemma** *Q-hit-pre-edges*: $edges\text{-of-walk } (Q\text{-hit-pre } @ [y]) \cap B \neq \{\}$ $\langle \text{proof} \rangle$ **lemma** *P-k-pre-edges*: $edges\text{-of-walk } (P\text{-k-pre } @ [y]) \cap B = \{\}$ $\langle \text{proof} \rangle$ **definition** *Q-hit'* **where** $Q\text{-hit}' \equiv P\text{-k-pre } @ y \# Q\text{-hit-post}$

private lemma *Q-hit'-v0-v1*: $v0 \rightsquigarrow Q\text{-hit}' \rightsquigarrow v1$ $\langle \text{proof} \rangle$ **lemma** *Q-hit'-v0-v1-H-x*: $v0 \rightsquigarrow Q\text{-hit}' \rightsquigarrow_{H-x}$ $v1$ $\langle \text{proof} \rangle$ **definition** *Q'* **where** $Q' \equiv insert\ Q\text{-hit}' (Q - \{Q\text{-hit}\})$

private lemma *Q-hit-edges-disjoint*:
 $\bigcup (edges\text{-of-walk } ' (Q - \{Q\text{-hit}\})) \cap edges\text{-of-walk } Q\text{-hit} = \{\}$
 $\langle \text{proof} \rangle$ **lemma** *Q-hit'-notin-Q-minus-Q-hit*: $Q\text{-hit}' \notin Q - \{Q\text{-hit}\}$ $\langle \text{proof} \rangle$ **lemma** *Q-weight-smaller*:
 $Q\text{-weight } Q' < Q\text{-weight } Q$ $\langle \text{proof} \rangle$ **lemma** *DisjointPaths-Q'*: $DisjointPaths\ H-x\ v0\ v1\ Q'$ $\langle \text{proof} \rangle$
lemma *card-Q'*: $card\ Q' = sep\text{-size}$ $\langle \text{proof} \rangle$

lemma *contradiction'*: $False$ $\langle \text{proof} \rangle$
end — anonymous context

corollary *contradiction*: $False$ $\langle \text{proof} \rangle$

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*:
 $\bigwedge paths' P\text{-new}'. ProofStepInduct\ G\ v0\ v1\ paths'\ P\text{-new}'\ sep\text{-size}$
 $\implies Digraph.distance (remove\text{-vertex } v0) (last\ P\text{-new}')\ v1$
 $\leq Digraph.distance (remove\text{-vertex } v0) (last\ P\text{-new}')\ v1$
begin

lemma *one-more-paths-exists-trivial*:
 $new\text{-last} = v1 \implies \exists paths. DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = Suc\ sep\text{-size}$
 $\langle \text{proof} \rangle$

lemma *one-more-paths-exists-nontrivial*:
assumes $new\text{-last} \neq v1$
shows $\exists paths. DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = Suc\ sep\text{-size}$
 $\langle \text{proof} \rangle$

corollary *one-more-paths-exists*:
shows $\exists paths. DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = Suc\ sep\text{-size}$
 $\langle \text{proof} \rangle$

end

lemma (**in** *ProofStepInduct*) *one-more-paths-exists*:
 $\exists paths. DisjointPaths\ G\ v0\ v1\ paths \wedge card\ paths = Suc\ sep\text{-size}$
 $\langle \text{proof} \rangle$

11.1 Menger's Theorem

theorem (in *v0-v1-Digraph*) *menger*:

assumes $\bigwedge S. \text{Separation } G \ v0 \ v1 \ S \implies \text{card } S \geq n$

shows $\exists \text{paths}. \text{DisjointPaths } G \ v0 \ v1 \ \text{paths} \wedge \text{card paths} = n$

<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 $\text{DisjointPaths } G \ v0 \ v1 \ \text{paths} \ \text{card paths} = n$

shows $\bigwedge S. \text{Separation } G \ v0 \ v1 \ S \implies \text{card } S \geq n$

<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** $n :: \text{nat}$

assumes $v0-V: v0 \in V$

and $v1-V: v1 \in V$

and $v0\text{-nonadj-}v1: \neg v0 \rightarrow v1$

and $v0\text{-neq-}v1: v0 \neq v1$

and $\text{no-small-separators}: \bigwedge S.$

$\llbracket S \subseteq V; v0 \notin S; v1 \notin S; \bigwedge xs. v0 \rightsquigarrow xs \rightsquigarrow v1 \implies \text{set } xs \cap S \neq \{\} \rrbracket \implies \text{card } S \geq n$

shows $\exists \text{paths}. \text{card paths} = n \wedge (\forall xs \in \text{paths}.$

$v0 \rightsquigarrow xs \rightsquigarrow v1 \wedge (\forall ys \in \text{paths} - \{xs\}. (\forall v \in \text{set } xs \cap \text{set } ys. v = v0 \vee v = v1)))$

<proof>

theorem (in *Digraph*) *menger-trivial*:

fixes $v0 \ v1 :: 'a$ **and** $n :: \text{nat}$

assumes $v0-V: v0 \in V$

and $v1-V: v1 \in V$

and $v0\text{-nonadj-}v1: \neg v0 \rightarrow v1$

and $v0\text{-neq-}v1: v0 \neq v1$

and $n\text{-paths}: \text{card paths} = n$

and $\text{paths-disjoint}: \forall xs \in \text{paths}.$

$v0 \rightsquigarrow xs \rightsquigarrow v1 \wedge (\forall ys \in \text{paths} - \{xs\}. (\forall v \in \text{set } xs \cap \text{set } ys. v = v0 \vee v = v1))$

shows $\bigwedge S. \llbracket S \subseteq V; v0 \notin S; v1 \notin S; \bigwedge xs. v0 \rightsquigarrow xs \rightsquigarrow v1 \implies \text{set } xs \cap S \neq \{\} \rrbracket \implies \text{card } S \geq n$

<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>.