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.
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.2</td>
<td>The Last Vertex of the New Path</td>
<td>14</td>
</tr>
<tr>
<td>7.3</td>
<td>Removing the Last Vertex</td>
<td>15</td>
</tr>
<tr>
<td>7.4</td>
<td>A New Path Following the Other Paths</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>Induction of Menger’s Theorem</td>
<td>16</td>
</tr>
<tr>
<td>8.1</td>
<td>No Small Separations</td>
<td>16</td>
</tr>
<tr>
<td>8.2</td>
<td>Choosing Paths Avoiding (\text{new_last})</td>
<td>16</td>
</tr>
<tr>
<td>8.3</td>
<td>Finding a Path Avoiding (Q)</td>
<td>17</td>
</tr>
<tr>
<td>8.4</td>
<td>Decomposing (P_k)</td>
<td>18</td>
</tr>
<tr>
<td>9</td>
<td>The case (y = \text{new_last})</td>
<td>18</td>
</tr>
<tr>
<td>10</td>
<td>The case (y \neq \text{new_last})</td>
<td>20</td>
</tr>
<tr>
<td>11</td>
<td>Menger’s Theorem</td>
<td>21</td>
</tr>
<tr>
<td>11.1</td>
<td>Menger’s Theorem</td>
<td>22</td>
</tr>
<tr>
<td>11.2</td>
<td>Self-contained Statement of the Main Theorem</td>
<td>22</td>
</tr>
<tr>
<td>Bibliography</td>
<td></td>
<td>23</td>
</tr>
</tbody>
</table>
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** `∃ b ∈ B. b ∉ 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 ∪ C B ∩ C = {}`
- **Shows** `card (A ∩ X) = card (B ∩ X) + card (C ∩ 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 ∈ set xs; x ≠ last xs ] ⇒ x ∈ 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** `∃ x. P x`
- **Obtains** `x` where `P x ∧ y. P y ⇒ weight x ≤ weight y`

**Proof**

### 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** `Edge = 'a × 'a`

**Type-synonym** `Walk = 'a list`

**Record** `Graph = verts : 'a set (V_1)`

**Record** `arcs : 'a Edge set (E_1)`

**Abbreviation** `is-arc :: ('a, 'b) Graph-scheme ⇒ 'a ⇒ 'a ⇒ bool (infixl → 60)`

**Abbreviation** `v →_G w ≡ (v,w) ∈ 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 ⊆ V × V 

context Digraph begin

lemma finite-edge-set [simp]: finite E 
⟨proof⟩

lemma edges-are-in-V: assumes v→w shows v ∈ V w ∈ 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 ∈ V ⇒ walk [v] 
  | Cons: v→w ⇒ walk (w # vs) ⇒ 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→w ⇒ 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)
We introduce shorthand notation for a path connecting two vertices.

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 4.3: The Set of All Paths

Definition all-paths where all-paths ≡ { 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 ⇒ ‘a Walk ⇒ ‘a ⇒ bool

(path-from-to G v xs w ≡ Digraph.path G xs ∧ xs ≠ Nil ∧ hd xs = v ∧ last xs = w)

Context Digraph begin

Lemma path-from-toI [intro]: [ path xs; xs ≠ Nil; hd xs = v; last xs = w ] ⇒ v ~ xs ~ w

and path-from-toE [dest]: v ~ xs ~ w ⇒ path xs ∧ xs ≠ Nil ∧ hd xs = v ∧ last xs = w

(proof)

Lemma path-from-to-ends: v ~ (xs @ w # ys) ~ w ⇒ ys = Nil

(proof)

Lemma path-from-to-combine:

assumes v ~ (xs @ x # xs') ~ w w' ~ (ys @ x # ys') ~ w' set xs ∩ set ys' = {}

shows v ~ (xs @ x # ys') ~ w'

(proof)

Lemma path-from-to-first: v ~ xs ~ w ⇒ v ∉ set (tl xs)

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

\textit{proof}

lemma path-from-to-last: \( v \leadsto xs \leadsto w \implies w \notin \text{set } (\text{butlast } xs) \)

\textit{proof}

lemma path-from-to-last': \( v \leadsto (xs @ x # xs') \leadsto w \implies w \notin \text{set } xs \)

\textit{proof}

Every walk contains a path connecting the same vertices.

lemma walk-to-path:
\begin{itemize}
\item \textbf{assumes} \( \text{walk } xs xs \neq \text{Nil} \) \( \text{hd } xs = v \text{ last } xs = w \)
\item \textbf{shows} \( \exists \text{ ys. } v \leadsto ys \leadsto w \land \text{set } ys \subseteq \text{set } xs \)
\end{itemize}

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

lemma edges-of-walkE: \((v, w) \in \text{edges-of-walk } xs \implies \exists \text{xs-pre } xs\text{-post. } xs = xs\text{-pre } @ v \neq w \land xs\text{-post }\)

\textit{proof}

lemma edges-of-walk-in-E: \( \text{walk } xs \implies \text{edges-of-walk } xs \subseteq E \)

\textit{proof}

lemma edges-of-walk-finite: \( \text{walk } xs \implies \text{finite } (\text{edges-of-walk } xs) \)

\textit{proof}

lemma edges-of-walk-empty: \( \text{edges-of-walk } [] = \{ \} \text{ edges-of-walk } [v] = \{ \} \)

\textit{proof}

lemma edges-of-walk-2: \( \text{edges-of-walk } [v, w] = \{(v, w)\} \)

\textit{proof}

lemma edges-of-walk-edge: \( \text{ walk } xs; (v, w) \in \text{edges-of-walk } xs \implies v \leadsto w \)

\textit{proof}

lemma edges-of-walk-middle \([\text{simp}]\): \( (v, w) \in \text{edges-of-walk } (xs @ v \neq w \neq xs') \)

\textit{proof}

lemma edges-of-comp1: \( \text{edges-of-walk } xs \subseteq \text{edges-of-walk } (xs @ ys) \)

\textit{proof}

lemma edges-of-comp2: \( \text{edges-of-walk } ys \subseteq \text{edges-of-walk } (xs @ ys) \)

\textit{proof}

lemma walk-edges-decomp-simple:
\begin{itemize}
\item \( \text{edges-of-walk } (v \neq w \neq xs) = \{ (v, w) \} \cup \text{edges-of-walk } (w \neq xs) \) \( \text{(is A = B)} \)
\end{itemize}

\textit{proof}

lemma walk-edges-decomp:
\begin{itemize}
\item \( \text{edges-of-walk } (xs @ x \neq xs') = \text{edges-of-walk } (xs @ [x]) \cup \text{edges-of-walk } (x \neq xs') \)
\end{itemize}
(proof)

**Lemma walk-edges-decomp':**

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

(\text{proof})

**Lemma walk-edges-vertices:** assumes \((v, w) \in \text{edges-of-walk} \langle xs \rangle\) shows \(v \in \text{set} \langle xs \rangle\) \(w \in \text{set} \langle xs \rangle\)

(\text{proof})

**Lemma walk-edges-subset:**

assumes \(\text{edges-subsets}: \text{edges-of-walk} \langle xs \rangle \subseteq \text{edges-of-walk} \langle ys \rangle\) and non-trivial: \(\text{tl} \langle xs \rangle \neq \text{Nil}\)

shows \(\text{set} \langle xs \rangle \subseteq \text{set} \langle ys \rangle\)

(\text{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 \(\text{path} \langle xs \rangle (v,w) \in \text{edges-of-walk} \langle xs \rangle\)

shows \(\exists \langle xs \rangle - \text{pre} \langle xs \rangle - \text{post}, \langle xs \rangle = \langle xs \rangle - \text{pre} \# v \# w \# \langle xs \rangle - \text{post}\)

\(\land (v,w) \notin \text{edges-of-walk} \langle \langle xs \rangle - \text{pre} \# [v] \rangle\)

\(\land (v,w) \notin \text{edges-of-walk} \langle \langle w \# \langle xs \rangle - \text{post} \rangle\rangle\)

(\text{proof})

**Lemma path-edges-remove-prefix:**

assumes \(\text{path} \langle xs @ x \# \langle xs \rangle' \rangle\)

shows \(\text{edges-of-walk} \langle \langle xs @ [x] \rangle\rangle = \text{edges-of-walk} \langle \langle xs @ [x] \rangle\rangle - \text{edges-of-walk} \langle \langle x \# \langle xs \rangle' \rangle\rangle\)

(\text{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

\(\text{first-edge-of-walk} (v \# w \# \langle xs \rangle) = (v, w)\)

\(\mid \text{first-edge-of-walk} [v] = \text{undefined}\)

\(\mid \text{first-edge-of-walk} [] = \text{undefined}\)

**Lemma first-edge-in-edges:** \(\text{tl} \langle xs \rangle \neq \text{Nil} \implies \text{first-edge-of-walk} \langle xs \rangle \in \text{edges-of-walk} \langle xs \rangle\)

(\text{proof})

**Lemma first-edge-hd-tl:** \([ v \sim \langle xs \rangle \mapsto w; \text{tl} \langle xs \rangle \neq \text{Nil} \]\(\implies \text{first-edge-of-walk} \langle xs \rangle = (v, \text{hd} \langle \text{tl} \langle xs \rangle \rangle)\)

(\text{proof})

**Lemma first-edge-first:**

assumes \(v \sim \langle xs \rangle \mapsto w (v,w') \in \text{edges-of-walk} \langle xs \rangle\)

shows \(\text{first-edge-of-walk} \langle xs \rangle = (v,w')\)

(\text{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: \(\forall xs'. v→xs'→w \Rightarrow length xs ≤ length xs'\)
\[shows\] 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
\[remove-vertex x \equiv G\{ verts := V - \{x\}, arcs := \text{Restr } E \{ V - \{x\} \}\}\]

lemma remove-vertex-V: Vremove-vertex x = V - \{x\} ⟨proof⟩
lemma remove-vertex-V!: Vremove-vertex x ⊆ V ⟨proof⟩
lemma remove-vertex-E: Eremove-vertex x = \text{Restr } E \{ V - \{x\} \} ⟨proof⟩
lemma remove-vertex-E": v→remove-vertex x w ⇒ v→w ⟨proof⟩
lemma remove-vertex-E'": \[ v→w; v \neq x; w \neq x \] ⇒ 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:
\[ [v \leadsto xs \leadsto w; x \in V; x \notin \text{set } xs] \implies v \leadsto xs \leadsto \text{remove-vertex } x \ w \]
\(\langle \text{proof} \rangle\)
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 \((\text{remove-vertex } x) \ xs\)
shows walk \(xs\)
\(\langle \text{proof} \rangle\)

lemma remove-vertex-path-from-to-add:
\[ v \leadsto xs \leadsto \text{remove-vertex } x \ w \implies v \leadsto xs \leadsto w \]
\(\langle \text{proof} \rangle\)

end — context Digraph

4.8 Two Distinguished Distinct Non-adjacent Vertices.

The setup for Menger’s Theorem requires two distinguished distinct non-adjacent vertices \(v_0\) and \(v_1\). Let us pin down this concept with the following locale.

locale \(v_0-v_1\text{-Digraph} = \text{Digraph} \ +\)
\(\text{fixes } v_0 \ v_1 :: 'a\)
\(\text{assumes } v_0\text{-V: } v_0 \in V \ \text{and } v_1\text{-V: } v_1 \in V\)
\(\text{and } v_0\text{-nonadj-v1: } \neg v_0 \to v_1\)
\(\text{and } v_0\text{-neq-v1: } v_0 \neq v_1\)
The only lemma we need about \(v_0-v_1\text{-Digraph}\) for now is that it is closed under removing a vertex that is not \(v_0\) or \(v_1\).

lemma \((\text{in } v_0-v_1\text{-Digraph}) \text{ remove-vertices-v0-v1-Digraph}:\)
\(\text{assumes } v \neq v_0 \ v \neq v_1\)
\(\text{shows } v_0\text{-v1-Digraph } (\text{remove-vertex } v) \ v_0 \ v_1\)
\(\langle \text{proof} \rangle\)

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 +
\(\text{assumes undirected: } v \to w = w \to v\)
\(\text{and no-loops: } \neg v \to v\)

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 ⊆ V
  and v0-notin-S: v0 ∉ S
  and v1-notin-S: v1 ∉ S
  and S-separates: \( \forall xs. v0 \leadsto xs \leadsto v1 \implies \text{set } xs \cap S \neq \{ \} \)

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

lemma (in v0-v1-Digraph) subgraph-separation-extend:
  assumes v ≠ v0 v ≠ v1 v ∈ 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 ≠ v0 v ≠ v1 v ∈ V
  and no-small-separation: \( \forall S. \text{Separation } G v0 v1 S \implies \text{card } S \geq \text{Suc } n \)
  and Separation (remove-vertex v) v0 v1 S
  shows card S ≥ n ⟨proof⟩

lemma (in v0-v1-Digraph) path-exists-if-no-separation:
  assumes S ⊆ V v0 \notin S v1 \notin S \neg \text{Separation } G v0 v1 S
  shows \( \exists xs. v0 \leadsto xs \leadsto v1 \land \text{set } xs \cap S = \{ \} \) ⟨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:
\[ \wedge xs. \, xs \in \text{paths} \implies v_0 \sim xs \sim v_1 \]
and paths-disjoint: \[ \wedge xs \, v. \, [ \, xs \in \text{paths}; \, xs \neq P; \, v \in \set{xs}; \, v \in \set{P} \,] \implies v = v_0 \lor v = v_1 \]

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 \setminus paths)) (\proof)

**lemma** paths-tl-not-nil: 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: \[ \, \, xs \in \text{paths}; \, ys \in \text{paths}; \, xs \neq ys \] \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 \sim P \sim v1
- and P-disjoint: \[ \wedge xs \, v. \, [ \, xs \in \text{paths}; \, xs \neq P; \, v \in \set{xs}; \, v \in \set{P} \,] \implies v = v_0 \lor v = v_1 \]
- shows DisjointPaths G v0 v1 (insert P 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 :: Walk. hd (tl xs)
definition second-vertices where second-vertices ≡ second-vertex ' paths

lemma second-vertex-inj: inj-on second-vertex paths

lemma second-vertices-card: card second-vertices = card paths

lemma second-vertices-in-V: second-vertices ⊆ V

lemma v0-v1-notin-second-vertices: v0 /∈ second-vertices v1 /∈ second-vertices

lemma second-vertices-new-path: hd (tl xs) /∈ second-vertices ⇒ xs /∈ paths

lemma second-vertices-first-edge: 
  \[ \begin{array}{c} xs \in paths; \ first-edge-of-walk xs = (v,w) \end{array} \] \implies w ∈ second-vertices

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:
  \[ \begin{array}{c} no-small-separations; S. \ Separation G v0 v1 S \implies card S ≥ Suc (card paths) \end{array} \] 
  \[ \exists P-new. v0 \not{\rightarrow}P-new\not{\rightarrow}v1 \land set P-new \cap second-vertices = \{\} \] 

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 ≡ \lambda x. x \neq v0 \land (\exists xs \in paths. x \in set xs) \lor x = v1

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 \( v_1 \), in which case we are already done because we have found a new disjoint path between \( v_0 \) and \( v_1 \).

locale DisjointPathsPlusOne = DisjointPaths +
  fixes P-new :: 'a Walk
  assumes P-new:
  \( v_0 \leadsto P\text{-}new \leadsto (\text{last P-new}) \)
  and tl-P-new:
  \( \text{tl P-new} \neq \text{Nil} \)
  \( \text{hd (tl P-new)} \notin \text{second-vertices} \)
  and last-P-new:
  \( \text{hitting-paths (last P-new)} \)
  \( \forall v. v \in \text{set (butlast P-new)} \implies \neg \text{hitting-paths v} \)

begin

7.1 Characterizing the New Path

lemma P-new-hd-disjoint: \( \forall xs. xs \in \text{paths} \implies \text{hd (tl P-new)} \neq \text{hd (tl xs)} \)
  ⟨proof⟩

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

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

lemma card-paths-with-new: \( \text{card paths-with-new} = \text{Suc (card paths)} \)
  ⟨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: \( xs \in \text{paths-with-new} \implies \text{hd xs} = v_0 \)
  ⟨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 \( \text{new-last} \).

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

definition new-last where new-last \equiv \text{last P-new}

lemma P-new-decomp: P-new = new-pre @ [new-last]
  ⟨proof⟩

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

lemma new-pre-hitting: \( x' \in \text{set new-pre} \implies \neg \text{hitting-paths x'} \)
  ⟨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 ~ R ~ 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 \( P \in \text{path-with-new} \). 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 ~ w tl xs ≠ Nil v1 ≠ set xs new-last ≠ set xs
  and P: P ∈ path-with-new hd (tl xs) = hd (tl P)
  and edges-subset: edges-of-walk xs ⊆ \( \bigcup \) (edges-of-walk \( ' \) path-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: S. Separation G v0 v1 S \implies\ card S \geq Suc sep-size
— The induction hypothesis.
  and no-small-separations-hyp: G'. (S. Separation G' v0 v1 S \implies\ card S \geq sep-size)
\implies\ v0-v1-Digraph G' v0 v1
\implies\ \exists\ paths. DisjointPaths G' v0 v1 paths \land\ 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 \( \bigcup (\text{edges-of-walk } ' Q) \cap (E - \bigcup (\text{edges-of-walk } ' \text{paths-with-new})) \) is minimal.

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

definition Q-weight where Q-weight ≡ λ Q. card (\( \bigcup (\text{edges-of-walk } ' Q) \cap B \))

definition Q-good where Q-good ≡ λ Q. DisjointPaths H-x v0 v1 Q ∧ card Q = sep-size ∧ (\( \forall Q'. \text{DisjointPaths H-x v0 v1 Q'} \land card Q' = sep-size \implies Q-weight Q \leq Q-weight Q' \))

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: \( \forall Q'. \text{DisjointPaths H-x v0 v1 Q'} \land card Q' = sep-size \implies Q-weight Q \leq Q-weight Q' \)
(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 \textit{new-last}, let $y$ be the first such vertex on $P_k$. Then we can split $P_k$ at this vertex.

\begin{verbatim}
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: $\forall y'$. $y' \in$ set $P_k$-pre $\implies$ ¬hitting-$Q$-or-$new$-last $y'$
\end{verbatim}

We can always go from $\text{ProofStepInduct-NonTrivial}$ to $\text{ProofStepInduct-NonTrivial-P-k-pre}$.

\begin{verbatim}
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
\end{verbatim}

\begin{verbatim}
context ProofStepInduct-NonTrivial-P-k-pre begin
  lemma $y$-neq-$v0$: $y \neq v0$ (proof)

  lemma $P_k$-pre-not-Nil: $P_k$-pre $\neq$ Nil (proof)

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

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

  lemma $y$-eq-$v1$-implies-$P_k$-neq-$P$-new: assumes $y$ = $v1$
  shows $P_k$ $\neq$ $P$-new (proof)

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

  lemma $y$-eq-$v1$-solves:
      assumes $y$ = $v1$
      shows $\exists$ paths. DisjointPaths $G$ $v0$ $v1$ paths $\land$ card paths = Suc sep-size
      (proof)
  end — locale ProofStepInduct-NonTrivial-P-k-pre
end
\end{verbatim}

9 The case $y = \text{new}_\text{last}$

theory Y-eq-new-last imports MengerInduction begin

We may assume $y \neq v1$ now because $[\text{ProofStepInduct-NonTrivial-P-k-pre} \ ?G \ ?v0.0 \ ?v1.0 \ ?\text{paths} \ ?P$-new \ ?sep-size \ ?$P_k$-pre \ ?$y$ \ ?$P_k$-post; \ ?$y$ = ?$v1.0$] $\implies$ $\exists$ paths. DisjointPaths $G$
?v0.0 \ ?v1.0 \ ?paths \ ?\text{card paths} = \text{Suc} \ ?\text{sep-size}$ shows that $y = v1$ already gives us $\text{Suc}$ sep-size many disjoint paths.

We also assume that we have chosen the previous paths optimally in the sense that the distance from \textit{new-last} to $v1$ is minimal.
locale ProofStepInduct-g-eq-new-last = ProofStepInduct-NonTrivial-P-k-pre +
assumes y-neq-v1: y ≠ v1 and y-eq-new-last: y = new-last
and optimal-paths: \{\text{paths'} P-new'}.
ProofStepInduct G v0 v1 paths' P-new' sep-size
\implies H.\text{distance} (\text{last P-new}) v1 \leq H.\text{distance} (\text{last P-new'}) v1

begin

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

definition R where R = \{R-pre z R-post \mid R-pre \subseteq \text{set } \text{R-pre}'\}

lemma R: new-last \sim R \sim H v1 \land (\forall R'. new-last \sim R' \sim H v1 \implies \text{length } R \leq \text{length } R')

lemma v1-in-Q: \exists Q-hit \in Q. v1 \in \text{set } Q-hit

lemma R-hits-Q: \exists z \in \text{set } R. Q.\text{hitting-paths } z

lemma R-decomp-exists:
obtains R-pre z R-post
where R = R-pre @ z \# R-post
and z \in Q.\text{hitting-paths } z
and \exists z'. z' \in \text{set } R-pre \implies \neg Q.\text{hitting-paths } z'

\langle \text{proof} \rangle

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 \in Q.\text{hitting-paths } z
and z-min: \exists z'. z' \in \text{set } R-pre \implies \neg Q.\text{hitting-paths } z'

begin

private lemma z-neq-v0: z ≠ v0 \langle \text{proof} \rangle

lemma z-neq-new-last: z ≠ new-last \langle \text{proof} \rangle
lemma R-pre-neq-Nil: R-pre ≠ Nil \langle \text{proof} \rangle
lemma z-closer-than-new-last: H.\text{distance } z v1 < H.\text{distance } new-last v1 \langle \text{proof} \rangle
definition R'-walk where R'-walk = P-k-pre @ R-pre @ [z]

private lemma R'-walk-not-Nil: R'-walk ≠ Nil \langle \text{proof} \rangle
lemma R'-walk-no-Q: \forall v \in \text{set } R'-walk. v ≠ z \implies \neg Q.\text{hitting-paths } v

\langle \text{proof} \rangle

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 ≠ ]; hd ?xs = ?v; last ?xs = ?w] \implies \exists ys. ?v ∪ ys ∪ ?w ∧ set ys
\subseteq set ?xs to transform R'-walk into a path R'.

private definition R' where
R' = \{R'. hd (tl R'-walk) ∪ R'-walk z \mid set R' \subseteq set (tl R'-walk)\}
private lemma R': hd (tl R'-walk) ⾃ R' ⋐ z set R' ⊆ set (tl R'-walk) ⟨proof⟩ lemma hd-R': hd R' = hd (tl P-k) ⟨proof⟩ lemma R'-no-Q: [ v ∈ set R'; v ≠ z ] → ¬Q.hitting-paths v ⟨proof⟩ lemma v0-R'-path: v0 ⾃ (v0 ≠ R') ⋐ z ⟨proof⟩ lemma z-eq-v1-solves: assumes z = v1 shows ∃ paths. DisjointPaths G v0 v1 paths ∧ card paths = Suc sep-size ⟨proof⟩ lemma z-neq-v1-solves: assumes z ≠ v1 shows ∃ paths. DisjointPaths G v0 v1 paths ∧ card paths = Suc sep-size ⟨proof⟩
corollary with-optimal-paths-solves': shows ∃ paths. DisjointPaths G v0 v1 paths ∧ card paths = Suc sep-size ⟨proof⟩
end — anonymous context
corollary with-optimal-paths-solves: ∃ paths. DisjointPaths G v0 v1 paths ∧ card paths = Suc sep-size ⟨proof⟩
end — locale ProofStepInduct-y-eq-new-last
dend

10 The case \( y \neq \text{new\_last} \)

theory Y-neq-new-last imports MengerInduction begin

Let us now consider the case that \( y \neq v1 \text{ and } y \neq \text{new\_last} \). Our goal is to show that this is inconsistent: The following locale will be unsatisfiable, proving that \( y = v1 \text{ or } y = \text{new\_last} \) holds.

locale ProofStepInduct-y-neq-new-last = ProofStepInduct-NonTrivial-P-k-pre +
assumes y-neq-v1: y ≠ v1 and y-neq-new-last: y ≠ new-last begin

lemma Q-hit-exists: obtains Q-hit Q-hit-pre Q-hit-post where
Q-hit ∈ Q y ∈ set Q-hit = Q-hit-pre @ y ≠ Q-hit-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-hit ∈ Q y ∈ set Q-hit
and Q-hit-decomp: Q-hit = Q-hit-pre @ y ≠ Q-hit-post
begin

private lemma Q-hit-v0-v1: v0 ⾃ Q-hit ⋐ v1 ⟨proof⟩ lemma Q-hit-vertices: set Q-hit ⊆ V − {new-last}
⟨proof⟩ lemma Q-hit-pre-not-Nil: Q-hit-pre ≠ Nil

20
(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 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** $\forall S. \text{Separation } G \text{ v0 v1 } S \Rightarrow \text{card } S \geq n$
- **Shows** $\exists \text{ paths. } \text{DisjointPaths } G \text{ v0 v1 paths } \land \text{card } \text{paths } = n$

(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** $\text{DisjointPaths } G \text{ v0 v1 paths } \land \text{card } \text{paths } = n$
- **Shows** $\forall S. \text{Separation } G \text{ v0 v1 } S \Rightarrow \text{card } S \geq n$

(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 $n\text{-small-separators: } \forall S. \begin{cases} S \subseteq V; v0 \notin S; v1 \notin S; \forall xs. v0 \xrightarrow{xs} v1 \Rightarrow \text{set } \text{xs } \cap S \neq \{} \end{cases} \Rightarrow \text{card } S \geq n$
- **Shows** $\exists \text{ paths. } \text{card } \text{paths } = n \land (\forall \text{xs } \in \text{paths}. (\forall v \in \text{set } \text{xs } \cap \text{set } v = v0 \lor v = v1)))$

(proof)

**Theorem (in Digraph) menger-trivial:**
- **Fixes** $v0 \ v1 :: 'a$ and $n :: \text{nat}$
- **Assumes** $v0\text{-V: } v0 \in V$
  - and $v1\text{-V: } v1 \in V$
  - and $v0\text{-nonadj-v1: } \neg v0 \rightarrow v1$
  - and $v0\text{-neq-v1: } v0 \neq v1$
  - and $n\text{-paths: } \text{card } \text{paths } = n$
  - and $n\text{-paths-disjoint: } \forall \text{xs } \in \text{paths}. v0 \xrightarrow{\text{xs}} v1 \land (\forall ys \in \text{paths} - \{\text{xs}\}. (\forall v \in \text{set } \text{xs } \cap \text{set } v = v0 \lor v = v1)))$
- **Shows** $\begin{cases} S \subseteq V; v0 \notin S; v1 \notin S; \forall xs. v0 \xrightarrow{xs} v1 \Rightarrow \text{set } \text{xs } \cap S \neq \{} \end{cases} \Rightarrow \text{card } S \geq n$

(proof)
References

