A formal proof of the max-flow min-cut theorem for countable networks

Andreas Lochbihler

September 13, 2023

Abstract

This article formalises a proof of the maximum-flow minimal-cut theorem for networks with countably many edges. A network is a directed graph with non-negative real-valued edge labels and two dedicated vertices, the source and the sink. A flow in a network assigns non-negative real numbers to the edges such that for all vertices except for the source and the sink, the sum of values on incoming edges equals the sum of values on outgoing edges. A cut is a subset of the vertices which contains the source, but not the sink. Our theorem states that in every network, there is a flow and a cut such that the flow saturates all the edges going out of the cut and is zero on all the incoming edges. The proof is based on the paper "The Max-Flow Min-Cut theorem for countable networks" by Aharoni et al. [2].

Additionally, we prove a characterisation of the lifting operation for relations on discrete probability distributions, which leads to a concise proof of its distributivity over relation composition.

Contents

1 Preliminaries

theory *MFMC-Misc* **imports**
  HOL−Probability.Probability
  HOL−Library.Transitive-Closure-Table
  HOL−Library.Complete-Partial-Order2
  HOL−Library.Bourbaki-Witt-Fixpoint
begin

hide-const (**open**) *cycle*
hide-const (**open**) *path*
hide-const (**open**) *cut*
hide-const (**open**) *orthogonal*

lemmas *disjE* [*consumes 1*, *case-names left right*, *cases pred*] = *disjE*

lemma *inj-on-Pair2* [*simp*]: *inj-on* (*Pair x*) *A*
⟨*proof*⟩

lemma *inj-on-Pair1* [*simp*]: *inj-on* ($\lambda x.$ (x, y)) *A*
⟨*proof*⟩

lemma *inj-map-prod'*: ⟦ *inj f*; *inj g* ⟧ $\Longrightarrow$ *inj-on* (*map-prod f g*) *X*
⟨*proof*⟩

lemma *not-range-Inr*: $x \notin$ *range Inr* $\longleftrightarrow$ $x \in$ *range Inl*
⟨*proof*⟩

lemma *not-range-Inl*: $x \notin$ *range Inl* $\longleftrightarrow$ $x \in$ *range Inr*
⟨*proof*⟩

lemma *Chains-into-chain*: $M \in$ *Chains* $\{(x, y).\ R\ x\ y\}$ $\Longrightarrow$ *Complete-Partial-Order.chain R M*
⟨*proof*⟩

lemma *chain-dual*: *Complete-Partial-Order.chain* ($\geq$) = *Complete-Partial-Order.chain* ($\leq$)
⟨*proof*⟩

lemma *Cauchy-real-Suc-diff*:
  fixes X :: *nat* $\Rightarrow$ *real* **and** x :: *real*
  assumes *bounded*: $\bigwedge n.\ |f\ (Suc\ n) - f\ n| \leq (c\ /\ x\ \hat{}\ n)$
  and x: $1 < x$
  shows *Cauchy f*
⟨*proof*⟩

lemma *complete-lattice-ccpo-dual*:
  class.ccpo Inf ($\geq$) (($>$) :: - :: *complete-lattice* $\Rightarrow$ -)
⟨*proof*⟩

lemma *card-eq-1-iff*: $card\ A = Suc\ 0 \longleftrightarrow (\exists x.\ A = \{x\})$
⟨*proof*⟩

lemma *nth-rotate1*: $n < length\ xs \Longrightarrow rotate1\ xs\ !\ n = xs\ !\ (Suc\ n\ mod\ length\ xs)$
⟨*proof*⟩

lemma *set-zip-rightI*: $[\![\ x \in set\ ys;\ length\ xs \geq length\ ys\ ]\!] \Longrightarrow \exists z.\ (z,\ x) \in set\ (zip\ xs\ ys)$
⟨*proof*⟩

lemma *map-eq-append-conv*:
$map\ f\ xs = ys\ @\ zs \longleftrightarrow (\exists ys'\ zs'.\ xs = ys'\ @\ zs' \wedge ys = map\ f\ ys' \wedge zs = map\ f\ zs')$
⟨*proof*⟩

lemma *rotate1-append*:
$rotate1\ (xs\ @\ ys) = (if\ xs = []\ then\ rotate1\ ys\ else\ tl\ xs\ @\ ys\ @\ [hd\ xs])$
⟨*proof*⟩

lemma *in-set-tlD*: $x \in set\ (tl\ xs) \Longrightarrow x \in set\ xs$
⟨*proof*⟩

lemma *countable-converseI*:
assumes *countable A*
shows *countable* (*converse A*)
⟨*proof*⟩

lemma *countable-converse* [*simp*]: $countable\ (converse\ A) \longleftrightarrow countable\ A$
⟨*proof*⟩

lemma *nn-integral-count-space-reindex*:
$inj\text{-}on\ f\ A \Longrightarrow (\int^{+} y.\ g\ y\ \partial count\text{-}space\ (f\ `\ A)) = (\int^{+} x.\ g\ (f\ x)\ \partial count\text{-}space\ A)$
⟨*proof*⟩

syntax
$\text{-}nn\text{-}sum :: pttrn \Rightarrow 'a\ set \Rightarrow 'b \Rightarrow 'b::comm\text{-}monoid\text{-}add\ ((2\sum^{+}\ \text{-}\in\text{-}./\ \text{-})\ [0,\ 51,\ 10]\ 10)$
$\text{-}nn\text{-}sum\text{-}UNIV :: pttrn \Rightarrow 'b \Rightarrow 'b::comm\text{-}monoid\text{-}add\ ((2\sum^{+}\ \text{-}./\ \text{-})\ [0,\ 10]\ 10)$
translations
$\sum^{+}\ i\in A.\ b \rightleftharpoons CONST\ nn\text{-}integral\ (CONST\ count\text{-}space\ A)\ (\lambda i.\ b)$
$\sum^{+}\ i.\ b \rightleftharpoons \sum^{+}\ i\in CONST\ UNIV.\ b$

inductive-simps *rtrancl-path-simps*:
$rtrancl\text{-}path\ R\ x\ []\ y$
$rtrancl\text{-}path\ R\ x\ (a\ \#\ bs)\ y$

definition $restrict\text{-}rel :: 'a\ set \Rightarrow ('a \times 'a)\ set \Rightarrow ('a \times 'a)\ set$
where $restrict\text{-}rel\ A\ R = \{(x,\ y)\in R.\ x \in A \wedge y \in A\}$

lemma *in-restrict-rel-iff*: $(x, y) \in restrict\text{-}rel\ A\ R \longleftrightarrow (x, y) \in R \wedge x \in A \wedge y \in A$
$\langle proof \rangle$

lemma *restrict-relE*: $[\![(x, y) \in restrict\text{-}rel\ A\ R; [\![(x, y) \in R; x \in A; y \in A]\!] \Longrightarrow thesis]\!] \Longrightarrow thesis$
$\langle proof \rangle$

lemma *restrict-relI* [*intro*!]: $[\![(x, y) \in R; x \in A; y \in A]\!] \Longrightarrow (x, y) \in restrict\text{-}rel\ A\ R$
$\langle proof \rangle$

lemma *Field-restrict-rel-subset*: $Field\ (restrict\text{-}rel\ A\ R) \subseteq A \cap Field\ R$
$\langle proof \rangle$

lemma *Field-restrict-rel* [*simp*]: $Refl\ R \Longrightarrow Field\ (restrict\text{-}rel\ A\ R) = A \cap Field\ R$
$\langle proof \rangle$

lemma *Partial-order-restrict-rel*:
  assumes *Partial-order R*
  shows *Partial-order* (*restrict-rel A R*)
$\langle proof \rangle$

lemma *Chains-restrict-relD*: $M \in Chains\ (restrict\text{-}rel\ A\ leq) \Longrightarrow M \in Chains\ leq$
$\langle proof \rangle$

lemma *bourbaki-witt-fixpoint-restrict-rel*:
  assumes *leq*: *Partial-order leq*
  and *chain-Field*: $\bigwedge M.\ [\![M \in Chains\ (restrict\text{-}rel\ A\ leq); M \neq \{\}]\!] \Longrightarrow lub\ M \in A$
  and *lub-least*: $\bigwedge M\ z.\ [\![M \in Chains\ leq; M \neq \{\}; \bigwedge x.\ x \in M \Longrightarrow (x, z) \in leq]\!] \Longrightarrow (lub\ M, z) \in leq$
  and *lub-upper*: $\bigwedge M\ z.\ [\![M \in Chains\ leq; z \in M]\!] \Longrightarrow (z, lub\ M) \in leq$
  and *increasing*: $\bigwedge x.\ [\![x \in A; x \in Field\ leq]\!] \Longrightarrow (x, f\ x) \in leq \wedge f\ x \in A$
  shows *bourbaki-witt-fixpoint lub* (*restrict-rel A leq*) *f*
$\langle proof \rangle$

lemma *Field-le* [*simp*]: $Field\ \{(x :: - :: preorder, y).\ x \leq y\} = UNIV$
$\langle proof \rangle$

lemma *Field-ge* [*simp*]: $Field\ \{(x :: - :: preorder, y).\ y \leq x\} = UNIV$
$\langle proof \rangle$

lemma *refl-le* [*simp*]: $refl\ \{(x :: - :: preorder, y).\ x \leq y\}$
$\langle proof \rangle$

lemma *refl-ge* [*simp*]: $refl\ \{(x :: - :: preorder, y).\ y \leq x\}$

⟨*proof*⟩

lemma *partial-order-le* [*simp*]: *partial-order-on UNIV* $\{(x :: - :: order, x').\ x \le x'\}$
⟨*proof*⟩

lemma *partial-order-ge* [*simp*]: *partial-order-on UNIV* $\{(x :: - :: order, x').\ x' \le x\}$
⟨*proof*⟩

lemma *incseq-chain-range*: *incseq f* $\Longrightarrow$ *Complete-Partial-Order.chain* ($\le$) (*range f*)
⟨*proof*⟩

end

theory *MFMC-Finite* **imports**
EdmondsKarp-Maxflow.EdmondsKarp-Termination-Abstract
HOL−Library.While-Combinator
begin

2 Existence of maximum flows and minimal cuts in finite graphs

This theory derives the existencs of a maximal flow or a minimal cut for finite graphs from the termination proof of the Edmonds-Karp algorithm.

context *Graph* **begin**

lemma *outgoing-outside*: $x \notin V \Longrightarrow$ *outgoing* $x = \{\}$
⟨*proof*⟩

lemma *incoming-outside*: $x \notin V \Longrightarrow$ *incoming* $x = \{\}$
⟨*proof*⟩

end

context *NFlow* **begin**

lemma *conservation*: $[\![\ x \ne s;\ x \ne t\ ]\!] \Longrightarrow$ *sum f* (*incoming x*) = *sum f* (*outgoing x*)
⟨*proof*⟩

lemma *augmenting-path-imp-shortest*:
isAugmentingPath p $\Longrightarrow \exists\, p.$ *Graph.isShortestPath cf s p t*
⟨*proof*⟩

lemma *shortest-is-augmenting*:
  Graph.isShortestPath cf s p t $\Longrightarrow$ *isAugmentingPath p*
  ⟨*proof*⟩

definition *augment-with-path p* $\equiv$ *augment* (*augmentingFlow p*)

end

context *Network* **begin**

definition *shortest-augmenting-path f* = (*SOME p. Graph.isShortestPath* (*residualGraph c f*) *s p t*)

lemma *shortest-augmenting-path*:
  assumes *NFlow c s t f*
  and $\exists$ *p. NPreflow.isAugmentingPath c s t f p*
  shows *Graph.isShortestPath* (*residualGraph c f*) *s* (*shortest-augmenting-path f*) *t*
  ⟨*proof*⟩

definition *max-flow* **where**
  max-flow = *while*
    (λ*f*. $\exists$ *p. NPreflow.isAugmentingPath c s t f p*)
    (λ*f*. *NFlow.augment-with-path c f* (*shortest-augmenting-path f*)) (λ-. *0*)

lemma *max-flow*:
  NFlow c s t max-flow (**is** *?thesis1*)
  $\neg$ ($\exists$ *p. NPreflow.isAugmentingPath c s t max-flow p*) (**is** *?thesis2*)
⟨*proof*⟩

end

end

3 Matrices for given marginals

This theory derives from the finite max-flow min-cut theorem the existence of matrices with given marginals based on a proof by Georg Kellerer [4].

theory *Matrix-For-Marginals*
  imports *MFMC-Misc HOL−Library.Diagonal-Subsequence MFMC-Finite*
begin

lemma *bounded-matrix-for-marginals-finite*:
  fixes *f g* :: *nat* $\Rightarrow$ *real*
    and *n* :: *nat*
    and *R* :: (*nat* $\times$ *nat*) *set*
  assumes *eq-sum*: *sum f* {..*n*} = *sum g* {..*n*}
    and *le*: $\bigwedge$*X*. *X* $\subseteq$ {..*n*} $\Longrightarrow$ *sum f X* $\leq$ *sum g* (*R* `` *X*)

and *f-nonneg*: $\bigwedge x.\ 0 \leq f\ x$
and *g-nonneg*: $\bigwedge y.\ 0 \leq g\ y$
and *R*: $R \subseteq \{..n\} \times \{..n\}$
obtains $h :: nat \Rightarrow nat \Rightarrow real$
where $\bigwedge x\ y.\ [\![\ x \leq n;\ y \leq n\ ]\!] \Longrightarrow 0 \leq h\ x\ y$
and $\bigwedge x\ y.\ [\![\ 0 < h\ x\ y;\ x \leq n;\ y \leq n\ ]\!] \Longrightarrow (x,\ y) \in R$
and $\bigwedge x.\ x \leq n \Longrightarrow f\ x = sum\ (h\ x)\ \{..n\}$
and $\bigwedge y.\ y \leq n \Longrightarrow g\ y = sum\ (\lambda x.\ h\ x\ y)\ \{..n\}$
⟨*proof*⟩

lemma *convergent-bounded-family-nat*:
fixes $f :: nat \Rightarrow nat \Rightarrow real$
assumes *bounded*: $\bigwedge x.\ bounded\ (range\ (\lambda n.\ f\ n\ x))$
obtains k **where** $strict\text{-}mono\ k\ \bigwedge x.\ convergent\ (\lambda n.\ f\ (k\ n)\ x)$
⟨*proof*⟩

lemma *convergent-bounded-family*:
fixes $f :: nat \Rightarrow 'a \Rightarrow real$
assumes *bounded*: $\bigwedge x.\ x \in A \Longrightarrow bounded\ (range\ (\lambda n.\ f\ n\ x))$
and *A*: *countable A*
obtains k **where** $strict\text{-}mono\ k\ \bigwedge x.\ x \in A \Longrightarrow convergent\ (\lambda n.\ f\ (k\ n)\ x)$
⟨*proof*⟩

abbreviation $zero\text{-}on :: ('a \Rightarrow 'b :: zero) \Rightarrow 'a\ set \Rightarrow 'a \Rightarrow 'b$
where $zero\text{-}on\ f \equiv override\text{-}on\ f\ (\lambda\text{-}.\ 0)$

lemma *zero-on-le* [*simp*]: **fixes** $f :: 'a \Rightarrow 'b :: \{preorder,\ zero\}$ **shows**
$zero\text{-}on\ f\ X\ x \leq f\ x \longleftrightarrow (x \in X \longrightarrow 0 \leq f\ x)$
⟨*proof*⟩

lemma *zero-on-nonneg*: **fixes** $f :: 'a \Rightarrow 'b :: \{preorder,\ zero\}$ **shows**
$0 \leq zero\text{-}on\ f\ X\ x \longleftrightarrow (x \notin X \longrightarrow 0 \leq f\ x)$
⟨*proof*⟩

lemma *sums-zero-on*:
fixes $f :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
assumes *f*: *f sums s*
and *X*: *finite X*
shows $zero\text{-}on\ f\ X\ sums\ (s - sum\ f\ X)$
⟨*proof*⟩

lemma
fixes $f :: nat \Rightarrow 'a::real\text{-}normed\text{-}vector$
assumes *f*: *summable f*
and *X*: *finite X*
shows *summable-zero-on* [*simp*]: $summable\ (zero\text{-}on\ f\ X)$ (**is** *?thesis1*)
and *suminf-zero-on*: $suminf\ (zero\text{-}on\ f\ X) = suminf\ f - sum\ f\ X$ (**is** *?thesis2*)
⟨*proof*⟩

lemma *summable-zero-on-nonneg*:
fixes $f :: nat \Rightarrow 'a :: \{ordered\text{-}comm\text{-}monoid\text{-}add, linorder\text{-}topology, conditionally\text{-}complete\text{-}linorder\}$
assumes *f*: *summable f*
and *nonneg*: $\bigwedge x.\ 0 \le f\ x$
shows *summable* (*zero-on f X*)
⟨*proof*⟩

lemma *zero-on-ennreal* [*simp*]: *zero-on* ($\lambda x.$ *ennreal* (*f x*)) A = ($\lambda x.$ *ennreal* (*zero-on f A x*))
⟨*proof*⟩

lemma *sum-lessThan-conv-atMost-nat*:
fixes $f :: nat \Rightarrow 'b :: ab\text{-}group\text{-}add$
shows $sum\ f\ \{..<n\} = sum\ f\ \{..n\} - f\ n$
⟨*proof*⟩

lemma *Collect-disjoint-atLeast*:
$Collect\ P \cap \{x..\} = \{\} \longleftrightarrow (\forall\, y \ge x.\ \neg\ P\ y)$
⟨*proof*⟩

lemma *bounded-matrix-for-marginals-nat*:
fixes $f\ g :: nat \Rightarrow real$
and $R :: (nat \times nat)\ set$
and $s :: real$
assumes *sum-f*: *f sums s* **and** *sum-g*: *g sums s*
and *f-nonneg*: $\bigwedge x.\ 0 \le f\ x$ **and** *g-nonneg*: $\bigwedge y.\ 0 \le g\ y$
and *f-le-g*: $\bigwedge X.\ suminf\ (zero\text{-}on\ f\ (-\ X)) \le suminf\ (zero\text{-}on\ g\ (-\ R\ ``\ X))$
obtains $h :: nat \Rightarrow nat \Rightarrow real$
where $\bigwedge x\ y.\ 0 \le h\ x\ y$
and $\bigwedge x\ y.\ 0 < h\ x\ y \Longrightarrow (x,\ y) \in R$
and $\bigwedge x.\ h\ x$ *sums* $f\ x$
and $\bigwedge y.\ (\lambda x.\ h\ x\ y)$ *sums* $g\ y$
⟨*proof*⟩

lemma *bounded-matrix-for-marginals-ennreal*:
assumes *sum-eq*: $(\sum^+ x{\in}A.\ f\ x) = (\sum^+ y{\in}B.\ g\ y)$
and *finite*: $(\sum^+ x{\in}B.\ g\ x) \ne \top$
and *le*: $\bigwedge X.\ X \subseteq A \Longrightarrow (\sum^+ x{\in}X.\ f\ x) \le (\sum^+ y{\in}R\ ``\ X.\ g\ y)$
and *countable* [*simp*]: *countable A countable B*
and *R*: $R \subseteq A \times B$
obtains *h* **where** $\bigwedge x\ y.\ 0 < h\ x\ y \Longrightarrow (x,\ y) \in R$
and $\bigwedge x\ y.\ h\ x\ y \ne \top$
and $\bigwedge x.\ x \in A \Longrightarrow (\sum^+ y{\in}B.\ h\ x\ y) = f\ x$
and $\bigwedge y.\ y \in B \Longrightarrow (\sum^+ x{\in}A.\ h\ x\ y) = g\ y$
⟨*proof*⟩

end
theory *MFMC-Network* **imports**
MFMC-Misc

begin

4 Graphs

type-synonym $'v$ *edge* $= 'v \times 'v$

record $'v$ *graph* $=$
  edge $:: 'v \Rightarrow 'v \Rightarrow$ *bool*

abbreviation *edges* $:: ('v, 'more)$ *graph-scheme* $\Rightarrow 'v$ *edge set* $(\mathbf{E}\imath)$
where $\mathbf{E}_G \equiv \{(x, y).\ edge\ G\ x\ y\}$

definition *outgoing* $:: ('v, 'more)$ *graph-scheme* $\Rightarrow 'v \Rightarrow 'v$ *set* $(\mathbf{OUT}\imath)$
where $\mathbf{OUT}_G\ x = \{y.\ (x, y) \in \mathbf{E}_G\}$

definition *incoming* $:: ('v, 'more)$ *graph-scheme* $\Rightarrow 'v \Rightarrow 'v$ *set* $(\mathbf{IN}\imath)$
where $\mathbf{IN}_G\ y = \{x.\ (x, y) \in \mathbf{E}_G\}$

Vertices are implicitly defined as the endpoints of edges, so we do not allow isolated vertices. For the purpose of flows, this does not matter as isolated vertices cannot contribute to a flow. The advantage is that we do not need any invariant on graphs that the endpoints of edges are a subset of the vertices. Conversely, this design choice makes a few proofs about reductions on webs harder, because we have to adjust other sets which are supposed to be part of the vertices.

definition *vertex* $:: ('v, 'more)$ *graph-scheme* $\Rightarrow 'v \Rightarrow$ *bool*
where $vertex\ G\ x \longleftrightarrow Domainp\ (edge\ G)\ x \lor Rangep\ (edge\ G)\ x$

lemma *vertexI*:
  shows *vertexI1*: $edge\ \Gamma\ x\ y \Longrightarrow vertex\ \Gamma\ x$
  and *vertexI2*: $edge\ \Gamma\ x\ y \Longrightarrow vertex\ \Gamma\ y$
$\langle proof \rangle$

abbreviation *vertices* $:: ('v, 'more)$ *graph-scheme* $\Rightarrow 'v$ *set* $(\mathbf{V}\imath)$
where $\mathbf{V}_G \equiv Collect\ (vertex\ G)$

lemma V-*def*: $\mathbf{V}_G = fst\ `\ \mathbf{E}_G \cup snd\ `\ \mathbf{E}_G$
$\langle proof \rangle$

type-synonym $'v$ *path* $= 'v$ *list*

abbreviation *path* $:: ('v, 'more)$ *graph-scheme* $\Rightarrow 'v \Rightarrow 'v$ *path* $\Rightarrow 'v \Rightarrow$ *bool*
where $path\ G \equiv rtrancl\text{-}path\ (edge\ G)$

inductive *cycle* $:: ('v, 'more)$ *graph-scheme* $\Rightarrow 'v$ *path* $\Rightarrow$ *bool*
  for G
where — Cycles must not pass through the same node multiple times. Otherwise, the cycle might enter a node via two different edges and leave it via just one edge.

Thus, the clean-up lemma would not hold any more.
cycle: $\llbracket$ *path G v p v*; $p \neq []$; *distinct p* $\rrbracket \Longrightarrow$ *cycle G p*

inductive-simps *cycle-Nil* [*simp*]: *cycle G Nil*

abbreviation *cycles* :: (*'v*, *'more*) *graph-scheme* $\Rightarrow$ *'v path set*
where *cycles G* $\equiv$ *Collect* (*cycle G*)

lemma *countable-cycles* [*simp*]:
assumes *countable* ($\mathbf{V}_G$)
shows *countable* (*cycles G*)
⟨*proof*⟩

definition *cycle-edges* :: *'v path* $\Rightarrow$ *'v edge list*
where *cycle-edges p* = *zip p* (*rotate1 p*)

lemma *cycle-edges-not-Nil*: *cycle G p* $\Longrightarrow$ *cycle-edges p* $\neq []$
⟨*proof*⟩

lemma *distinct-cycle-edges*:
cycle G p $\Longrightarrow$ *distinct* (*cycle-edges p*)
⟨*proof*⟩

lemma *cycle-enter-leave-same*:
assumes *cycle G p*
shows *card* (*set* [(x', y) $\leftarrow$ *cycle-edges p*. $x' = x$]) = *card* (*set* [(x', y) $\leftarrow$ *cycle-edges p*. $y = x$])
(**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *cycle-leave-ex-enter*:
assumes *cycle G p* **and** (x, y) $\in$ *set* (*cycle-edges p*)
shows $\exists z$. (z, x) $\in$ *set* (*cycle-edges p*)
⟨*proof*⟩

lemma *cycle-edges-edges*:
assumes *cycle G p*
shows *set* (*cycle-edges p*) $\subseteq \mathbf{E}_G$
⟨*proof*⟩

5 Network and Flow

record *'v network* = *'v graph* +
capacity :: *'v edge* $\Rightarrow$ *ennreal*
source :: *'v*
sink :: *'v*

type-synonym *'v flow* = *'v edge* $\Rightarrow$ *ennreal*

inductive-set *support-flow* :: $'v$ *flow* $\Rightarrow$ $'v$ *edge set*
for *f*
where $f\ e > 0 \Longrightarrow e \in$ *support-flow f*

lemma *support-flow-conv*: *support-flow f* $= \{e.\ f\ e > 0\}$
⟨*proof*⟩

lemma *not-in-support-flowD*: $x \notin$ *support-flow f* $\Longrightarrow f\ x = 0$
⟨*proof*⟩

definition *d-OUT* :: $'v$ *flow* $\Rightarrow$ $'v \Rightarrow$ *ennreal*
where *d-OUT g x* $= (\sum^+\ y.\ g\ (x,\ y))$

definition *d-IN* :: $'v$ *flow* $\Rightarrow$ $'v \Rightarrow$ *ennreal*
where *d-IN g y* $= (\sum^+\ x.\ g\ (x,\ y))$

lemma *d-OUT-mono*: $(\bigwedge y.\ f\ (x,\ y) \leq g\ (x,\ y)) \Longrightarrow$ *d-OUT f x* $\leq$ *d-OUT g x*
⟨*proof*⟩

lemma *d-IN-mono*: $(\bigwedge x.\ f\ (x,\ y) \leq g\ (x,\ y)) \Longrightarrow$ *d-IN f y* $\leq$ *d-IN g y*
⟨*proof*⟩

lemma *d-OUT-0* [*simp*]: *d-OUT* $(\lambda$-. $0)\ x = 0$
⟨*proof*⟩

lemma *d-IN-0* [*simp*]: *d-IN* $(\lambda$-. $0)\ x = 0$
⟨*proof*⟩

lemma *d-OUT-add*: *d-OUT* $(\lambda e.\ f\ e + g\ e)\ x =$ *d-OUT f x* $+$ *d-OUT g x*
⟨*proof*⟩

lemma *d-IN-add*: *d-IN* $(\lambda e.\ f\ e + g\ e)\ x =$ *d-IN f x* $+$ *d-IN g x*
⟨*proof*⟩

lemma *d-OUT-cmult*: *d-OUT* $(\lambda e.\ c * f\ e)\ x = c *$ *d-OUT f x*
⟨*proof*⟩

lemma *d-IN-cmult*: *d-IN* $(\lambda e.\ c * f\ e)\ x = c *$ *d-IN f x*
⟨*proof*⟩

lemma *d-OUT-ge-point*: $f\ (x,\ y) \leq$ *d-OUT f x*
⟨*proof*⟩

lemma *d-IN-ge-point*: $f\ (y,\ x) \leq$ *d-IN f x*
⟨*proof*⟩

lemma *d-OUT-monotone-convergence-SUP*:
assumes *incseq* $(\lambda n\ y.\ f\ n\ (x,\ y))$
shows *d-OUT* $(\lambda e.$ *SUP* $n.\ f\ n\ e)\ x =$ (*SUP* $n.$ *d-OUT* $(f\ n)\ x)$

⟨*proof*⟩

lemma *d-IN-monotone-convergence-SUP*:
 assumes $incseq\ (\lambda n\ x.\ f\ n\ (x,\ y))$
 shows $d\text{-}IN\ (\lambda e.\ SUP\ n.\ f\ n\ e)\ y = (SUP\ n.\ d\text{-}IN\ (f\ n)\ y)$
⟨*proof*⟩

lemma *d-OUT-diff*:
 assumes $\bigwedge y.\ g\ (x,\ y) \leq f\ (x,\ y)\ d\text{-}OUT\ g\ x \neq \top$
 shows $d\text{-}OUT\ (\lambda e.\ f\ e - g\ e)\ x = d\text{-}OUT\ f\ x - d\text{-}OUT\ g\ x$
⟨*proof*⟩

lemma *d-IN-diff*:
 assumes $\bigwedge x.\ g\ (x,\ y) \leq f\ (x,\ y)\ d\text{-}IN\ g\ y \neq \top$
 shows $d\text{-}IN\ (\lambda e.\ f\ e - g\ e)\ y = d\text{-}IN\ f\ y - d\text{-}IN\ g\ y$
⟨*proof*⟩

lemma fixes G (**structure**)
 shows *d-OUT-alt-def*: $(\bigwedge y.\ (x,\ y) \notin \mathbf{E} \Longrightarrow g\ (x,\ y) = 0) \Longrightarrow d\text{-}OUT\ g\ x = (\sum^+\ y{\in}\mathbf{OUT}\ x.\ g\ (x,\ y))$
 and *d-IN-alt-def*: $(\bigwedge x.\ (x,\ y) \notin \mathbf{E} \Longrightarrow g\ (x,\ y) = 0) \Longrightarrow d\text{-}IN\ g\ y = (\sum^+\ x{\in}\mathbf{IN}\ y.\ g\ (x,\ y))$
⟨*proof*⟩

lemma *d-OUT-alt-def2*: $d\text{-}OUT\ g\ x = (\sum^+\ y{\in}\{y.\ (x,\ y) \in support\text{-}flow\ g\}.\ g\ (x,\ y))$
 and *d-IN-alt-def2*: $d\text{-}IN\ g\ y = (\sum^+\ x{\in}\{x.\ (x,\ y) \in support\text{-}flow\ g\}.\ g\ (x,\ y))$
⟨*proof*⟩

definition *d-diff* :: $('v\ edge \Rightarrow ennreal) \Rightarrow 'v \Rightarrow ennreal$
where $d\text{-}diff\ g\ x = d\text{-}OUT\ g\ x - d\text{-}IN\ g\ x$

abbreviation *KIR* :: $('v\ edge \Rightarrow ennreal) \Rightarrow 'v \Rightarrow bool$
where $KIR\ f\ x \equiv d\text{-}OUT\ f\ x = d\text{-}IN\ f\ x$

inductive-set *SINK* :: $('v\ edge \Rightarrow ennreal) \Rightarrow 'v\ set$
 for f
where *SINK*: $d\text{-}OUT\ f\ x = 0 \Longrightarrow x \in SINK\ f$

lemma *SINK-mono*:
 assumes $\bigwedge e.\ f\ e \leq g\ e$
 shows $SINK\ g \subseteq SINK\ f$
⟨*proof*⟩

lemma *SINK-mono'*: $f \leq g \Longrightarrow SINK\ g \subseteq SINK\ f$
⟨*proof*⟩

lemma *support-flow-Sup*: $support\text{-}flow\ (Sup\ Y) = (\bigcup f{\in}Y.\ support\text{-}flow\ f)$
⟨*proof*⟩

lemma
  assumes *chain*: *Complete-Partial-Order.chain* $(\leq)$ Y
  and Y: $Y \neq \{\}$
  and *countable*: *countable* (*support-flow* (*Sup* Y))
  shows *d-OUT-Sup*: *d-OUT* (*Sup* Y) x = (*SUP* $f \in Y$. *d-OUT* f x) (**is** *?OUT* x **is** *?lhs1* x = *?rhs1* x)
  and *d-IN-Sup*: *d-IN* (*Sup* Y) y = (*SUP* $f \in Y$. *d-IN* f y) (**is** *?IN* **is** *?lhs2* = *?rhs2*)
  and *SINK-Sup*: *SINK* (*Sup* Y) = ($\bigcap f \in Y$. *SINK* f) (**is** *?SINK*)
⟨*proof*⟩

lemma
  assumes *chain*: *Complete-Partial-Order.chain* $(\leq)$ Y
  and Y: $Y \neq \{\}$
  and *countable*: *countable* (*support-flow* f)
  and *bounded*: $\bigwedge g\ e.\ g \in Y \Longrightarrow g\ e \leq f\ e$
  shows *d-OUT-Inf*: *d-OUT* f $x \neq$ *top* $\Longrightarrow$ *d-OUT* (*Inf* Y) x = (*INF* $g \in Y$. *d-OUT* g x) (**is** - $\Longrightarrow$ *?OUT* **is** - $\Longrightarrow$ *?lhs1* = *?rhs1*)
  and *d-IN-Inf*: *d-IN* f $x \neq$ *top* $\Longrightarrow$ *d-IN* (*Inf* Y) x = (*INF* $g \in Y$. *d-IN* g x) (**is** - $\Longrightarrow$ *?IN* **is** - $\Longrightarrow$ *?lhs2* = *?rhs2*)
⟨*proof*⟩

inductive *flow* :: ('*v*, '*more*) *network-scheme* ⇒ '*v flow* ⇒ *bool*
  for Δ (**structure**) **and** f
where
  flow: ⟦ $\bigwedge e.\ f\ e \leq$ *capacity* Δ e;
    $\bigwedge x.$ ⟦ $x \neq$ *source* Δ; $x \neq$ *sink* Δ ⟧ $\Longrightarrow$ *KIR* f x ⟧
  $\Longrightarrow$ *flow* Δ f

lemma *flowD-capacity*: *flow* Δ $f \Longrightarrow f\ e \leq$ *capacity* Δ e
⟨*proof*⟩

lemma *flowD-KIR*: ⟦ *flow* Δ f; $x \neq$ *source* Δ; $x \neq$ *sink* Δ ⟧ $\Longrightarrow$ *KIR* f x
⟨*proof*⟩

lemma *flowD-capacity-OUT*: *flow* Δ $f \Longrightarrow$ *d-OUT* f $x \leq$ *d-OUT* (*capacity* Δ) x
⟨*proof*⟩

lemma *flowD-capacity-IN*: *flow* Δ $f \Longrightarrow$ *d-IN* f $x \leq$ *d-IN* (*capacity* Δ) x
⟨*proof*⟩

abbreviation *value-flow* :: ('*v*, '*more*) *network-scheme* ⇒ ('*v edge* ⇒ *ennreal*) ⇒ *ennreal*
where *value-flow* Δ f ≡ *d-OUT* f (*source* Δ)

5.1 Cut

type-synonym '*v cut* = '*v set*

inductive *cut* :: ($'v$, $'more$) *network-scheme* $\Rightarrow$ $'v$ *cut* $\Rightarrow$ *bool*
for Δ **and** S
where *cut*: $[\![$ *source* $\Delta \in S$; *sink* $\Delta \notin S$ $]\!] \Longrightarrow$ *cut* Δ S

inductive *orthogonal* :: ($'v$, $'more$) *network-scheme* $\Rightarrow$ $'v$ *flow* $\Rightarrow$ $'v$ *cut* $\Rightarrow$ *bool*
for Δ f S
where
$[\![$ $\bigwedge x\ y.$ $[\![$ *edge* Δ x y; $x \in S$; $y \notin S$ $]\!] \Longrightarrow f\ (x,\ y) =$ *capacity* Δ $(x,\ y)$;
$\bigwedge x\ y.$ $[\![$ *edge* Δ x y; $x \notin S$; $y \in S$ $]\!] \Longrightarrow f\ (x,\ y) = 0$ $]\!]$
$\Longrightarrow$ *orthogonal* Δ f S

lemma *orthogonalD-out*:
$[\![$ *orthogonal* Δ f S; *edge* Δ x y; $x \in S$; $y \notin S$ $]\!] \Longrightarrow f\ (x,\ y) =$ *capacity* Δ $(x,\ y)$
$\langle$*proof*$\rangle$

lemma *orthogonalD-in*:
$[\![$ *orthogonal* Δ f S; *edge* Δ x y; $x \notin S$; $y \in S$ $]\!] \Longrightarrow f\ (x,\ y) = 0$
$\langle$*proof*$\rangle$

5.2 Countable network

locale *countable-network* =
fixes Δ :: ($'v$, $'more$) *network-scheme* (**structure**)
assumes *countable-E* [*simp*]: *countable* **E**
and *source-neq-sink* [*simp*]: *source* $\Delta \neq$ *sink* Δ
and *capacity-outside*: $e \notin$ **E** $\Longrightarrow$ *capacity* Δ $e = 0$
and *capacity-finite* [*simp*]: *capacity* Δ $e \neq \top$
begin

lemma *sink-neq-source* [*simp*]: *sink* $\Delta \neq$ *source* Δ
$\langle$*proof*$\rangle$

lemma *countable-V* [*simp*]: *countable* **V**
$\langle$*proof*$\rangle$

lemma *flowD-outside*:
assumes g: *flow* Δ g
shows $e \notin$ **E** $\Longrightarrow g\ e = 0$
$\langle$*proof*$\rangle$

lemma *flowD-finite*:
assumes *flow* Δ g
shows $g\ e \neq \top$
$\langle$*proof*$\rangle$

lemma *zero-flow* [*simp*]: *flow* Δ (λ-. 0)
$\langle$*proof*$\rangle$

end

5.3 Reduction for avoiding antiparallel edges

locale *antiparallel-edges* = *countable-network* Δ
  for Δ :: (*'v*, *'more*) *network-scheme* (**structure**)
begin

We eliminate the assumption of antiparallel edges by adding a vertex for every edge. Thus, antiparallel edges are split up into a cycle of 4 edges. This idea already appears in [1].

datatype (*plugins del*: *transfer size*) *'v' vertex* = *Vertex 'v'* | *Edge 'v' 'v'*

inductive *edg* :: *'v vertex* $\Rightarrow$ *'v vertex* $\Rightarrow$ *bool*
where
  OUT: *edge* Δ *x y* $\Longrightarrow$ *edg* (*Vertex x*) (*Edge x y*)
| *IN*: *edge* Δ *x y* $\Longrightarrow$ *edg* (*Edge x y*) (*Vertex y*)

inductive-simps *edg-simps* [*simp*]:
  edg (*Vertex x*) *v*
  edg (*Edge x y*) *v*
  edg v (*Vertex x*)
  edg v (*Edge x y*)

fun *split* :: *'v flow* $\Rightarrow$ *'v vertex flow*
where
  split f (*Vertex x*, *Edge x' y*) = (*if x' = x then f* (*x*, *y*) *else 0*)
| *split f* (*Edge x y'*, *Vertex y*) = (*if y' = y then f* (*x*, *y*) *else 0*)
| *split f* - = *0*

lemma *split-Vertex1-eq-0I*: ($\bigwedge$*z*. $y \neq$ *Edge x z*) $\Longrightarrow$ *split f* (*Vertex x*, *y*) = *0*
⟨*proof*⟩

lemma *split-Vertex2-eq-0I*: ($\bigwedge$*z*. $y \neq$ *Edge z x*) $\Longrightarrow$ *split f* (*y*, *Vertex x*) = *0*
⟨*proof*⟩

lemma *split-Edge1-eq-0I*: ($\bigwedge$*z*. $y \neq$ *Vertex x*) $\Longrightarrow$ *split f* (*Edge z x*, *y*) = *0*
⟨*proof*⟩

lemma *split-Edge2-eq-0I*: ($\bigwedge$*z*. $y \neq$ *Vertex x*) $\Longrightarrow$ *split f* (*y*, *Edge x z*) = *0*
⟨*proof*⟩

definition Δ'' :: *'v vertex network*
where Δ'' = ⦇*edge* = *edg*, *capacity* = *split* (*capacity* Δ), *source* = *Vertex* (*source* Δ), *sink* = *Vertex* (*sink* Δ)⦈

lemma Δ''*-sel* [*simp*]:
  edge Δ'' = *edg*
  capacity Δ'' = *split* (*capacity* Δ)
  source Δ'' = *Vertex* (*source* Δ)
  sink Δ'' = *Vertex* (*sink* Δ)

$\langle proof \rangle$

lemma **E**-Δ'': $\mathbf{E}_{\Delta''} = (\lambda(x, y).\ (\mathit{Vertex}\ x,\ \mathit{Edge}\ x\ y))\ `\ \mathbf{E} \cup (\lambda(x, y).\ (\mathit{Edge}\ x\ y,\ \mathit{Vertex}\ y))\ `\ \mathbf{E}$
$\langle proof \rangle$

lemma **V**-Δ'': $\mathbf{V}_{\Delta''} = \mathit{Vertex}\ `\ \mathbf{V} \cup \mathit{case\text{-}prod}\ \mathit{Edge}\ `\ \mathbf{E}$
$\langle proof \rangle$

lemma *inj-on-Edge1* [*simp*]: $\mathit{inj\text{-}on}\ (\lambda x.\ \mathit{Edge}\ x\ y)\ A$
$\langle proof \rangle$

lemma *inj-on-Edge2* [*simp*]: $\mathit{inj\text{-}on}\ (\mathit{Edge}\ x)\ A$
$\langle proof \rangle$

lemma *d-IN-split-Vertex* [*simp*]: $\mathit{d\text{-}IN}\ (\mathit{split}\ f)\ (\mathit{Vertex}\ x) = \mathit{d\text{-}IN}\ f\ x$ (**is** *?lhs* = *?rhs*)
$\langle proof \rangle$

lemma *d-OUT-split-Vertex* [*simp*]: $\mathit{d\text{-}OUT}\ (\mathit{split}\ f)\ (\mathit{Vertex}\ x) = \mathit{d\text{-}OUT}\ f\ x$ (**is** *?lhs* = *?rhs*)
$\langle proof \rangle$

lemma *d-IN-split-Edge* [*simp*]: $\mathit{d\text{-}IN}\ (\mathit{split}\ f)\ (\mathit{Edge}\ x\ y) = \mathit{max}\ 0\ (f\ (x,\ y))$ (**is** *?lhs* = *?rhs*)
$\langle proof \rangle$

lemma *d-OUT-split-Edge* [*simp*]: $\mathit{d\text{-}OUT}\ (\mathit{split}\ f)\ (\mathit{Edge}\ x\ y) = \mathit{max}\ 0\ (f\ (x,\ y))$ (**is** *?lhs* = *?rhs*)
$\langle proof \rangle$

lemma Δ''-*countable-network*: *countable-network* Δ''
$\langle proof \rangle$

interpretation Δ'': *countable-network* Δ'' $\langle proof \rangle$

lemma *flow-split* [*simp*]:
  assumes *flow* Δ *f*
  shows *flow* Δ'' (*split f*)
$\langle proof \rangle$

abbreviation (*input*) *collect* :: $'v\ \mathit{vertex}\ \mathit{flow} \Rightarrow\ 'v\ \mathit{flow}$
where $\mathit{collect}\ f \equiv (\lambda(x, y).\ f\ (\mathit{Edge}\ x\ y,\ \mathit{Vertex}\ y))$

lemma *d-OUT-collect*:
  assumes *f*: *flow* Δ'' *f*
  shows $\mathit{d\text{-}OUT}\ (\mathit{collect}\ f)\ x = \mathit{d\text{-}OUT}\ f\ (\mathit{Vertex}\ x)$
$\langle proof \rangle$

lemma *flow-collect* [*simp*]:
  assumes *f*: *flow* Δ'' *f*
  shows *flow* Δ (*collect f*)
⟨*proof*⟩

lemma *value-collect*: *flow* Δ'' *f* $\Longrightarrow$ *value-flow* Δ (*collect f*) = *value-flow* Δ'' *f*
⟨*proof*⟩

end

end
theory *MFMC-Web* **imports**
  MFMC-Network
begin

6 Webs and currents

record *'v web* = *'v graph* +
  weight :: *'v* ⇒ *ennreal*
  A :: *'v set*
  B :: *'v set*

lemma *vertex-weight-update* [*simp*]: *vertex* (*weight-update f* Γ) = *vertex* Γ
⟨*proof*⟩

type-synonym *'v current* = *'v edge* ⇒ *ennreal*

inductive *current* :: (*'v*, *'more*) *web-scheme* ⇒ *'v current* ⇒ *bool*
  for Γ *f*
where
  current:
  $[\![\bigwedge x.\ d\text{-}OUT\ f\ x \leq weight\ \Gamma\ x;$
    $\bigwedge x.\ d\text{-}IN\ f\ x \leq weight\ \Gamma\ x;$
    $\bigwedge x.\ x \notin A\ \Gamma \Longrightarrow d\text{-}OUT\ f\ x \leq d\text{-}IN\ f\ x;$
    $\bigwedge a.\ a \in A\ \Gamma \Longrightarrow d\text{-}IN\ f\ a = 0;$
    $\bigwedge b.\ b \in B\ \Gamma \Longrightarrow d\text{-}OUT\ f\ b = 0;$
    $\bigwedge e.\ e \notin \mathbf{E}_\Gamma \Longrightarrow f\ e = 0\ ]\!]$
  $\Longrightarrow$ *current* Γ *f*

lemma *currentD-weight-OUT*: *current* Γ *f* $\Longrightarrow$ *d-OUT f x* ≤ *weight* Γ *x*
⟨*proof*⟩

lemma *currentD-weight-IN*: *current* Γ *f* $\Longrightarrow$ *d-IN f x* ≤ *weight* Γ *x*
⟨*proof*⟩

lemma *currentD-OUT-IN*: $[\![$ *current* Γ *f*; *x* ∉ *A* Γ $]\!]$ $\Longrightarrow$ *d-OUT f x* ≤ *d-IN f x*
⟨*proof*⟩

lemma *currentD-IN*: $[\![$ *current* Γ *f*; *a* ∈ *A* Γ $]\!]$ $\Longrightarrow$ *d-IN f a* = *0*

$\langle proof \rangle$

lemma *currentD-OUT*: $\llbracket$ *current* Γ f; $b \in B\ \Gamma$ $\rrbracket \Longrightarrow$ *d-OUT* f $b = 0$
$\langle proof \rangle$

lemma *currentD-outside*: $\llbracket$ *current* Γ f; $\neg$ *edge* Γ x y $\rrbracket \Longrightarrow f\ (x, y) = 0$
$\langle proof \rangle$

lemma *currentD-outside'*: $\llbracket$ *current* Γ f; $e \notin \mathbf{E}_\Gamma$ $\rrbracket \Longrightarrow f\ e = 0$
$\langle proof \rangle$

lemma *currentD-OUT-eq-0*:
  assumes *current* Γ f
  shows *d-OUT* f $x = 0 \longleftrightarrow (\forall\, y.\ f\ (x, y) = 0)$
$\langle proof \rangle$

lemma *currentD-IN-eq-0*:
  assumes *current* Γ f
  shows *d-IN* f $x = 0 \longleftrightarrow (\forall\, y.\ f\ (y, x) = 0)$
$\langle proof \rangle$

lemma *current-support-flow*:
  fixes Γ (**structure**)
  assumes *current* Γ f
  shows *support-flow* $f \subseteq \mathbf{E}$
$\langle proof \rangle$

lemma *currentD-outside-IN*: $\llbracket$ *current* Γ f; $x \notin \mathbf{V}_\Gamma$ $\rrbracket \Longrightarrow$ *d-IN* f $x = 0$
$\langle proof \rangle$

lemma *currentD-outside-OUT*: $\llbracket$ *current* Γ f; $x \notin \mathbf{V}_\Gamma$ $\rrbracket \Longrightarrow$ *d-OUT* f $x = 0$
$\langle proof \rangle$

lemma *currentD-weight-in*: *current* Γ $h \Longrightarrow h\ (x, y) \leq$ *weight* Γ y
  $\langle proof \rangle$

lemma *currentD-weight-out*: *current* Γ $h \Longrightarrow h\ (x, y) \leq$ *weight* Γ x
  $\langle proof \rangle$

lemma *current-leI*:
  fixes Γ (**structure**)
  assumes *f*: *current* Γ f
  and *le*: $\bigwedge e.\ g\ e \leq f\ e$
  and *OUT-IN*: $\bigwedge x.\ x \notin A\ \Gamma \Longrightarrow$ *d-OUT* g $x \leq$ *d-IN* g x
  shows *current* Γ g
$\langle proof \rangle$

lemma *current-weight-mono*:
  $\llbracket$ *current* Γ f; *edge* Γ = *edge* Γ'; $A\ \Gamma = A\ \Gamma'$; $B\ \Gamma = B\ \Gamma'$; $\bigwedge x.$ *weight* Γ $x \leq$

weight Γ' *x* $]\!]$
$\Longrightarrow$ *current* Γ' *f*
$\langle proof \rangle$

abbreviation (*input*) *zero-current* :: *'v current*
where *zero-current* $\equiv \lambda$-. *0*

lemma *SINK-0* [*simp*]: *SINK zero-current* = *UNIV*
$\langle proof \rangle$

lemma *current-0* [*simp*]: *current* Γ *zero-current*
$\langle proof \rangle$

inductive *web-flow* :: (*'v*, *'more*) *web-scheme* $\Rightarrow$ *'v current* $\Rightarrow$ *bool*
for Γ (**structure**) **and** *f*
where
web-flow: $[\![$ *current* Γ *f*; $\bigwedge x$. $[\![$ $x \in$ **V**; $x \notin A\ \Gamma$; $x \notin B\ \Gamma$ $]\!]$ $\Longrightarrow$ *KIR f x* $]\!]$ $\Longrightarrow$ *web-flow* Γ *f*

lemma *web-flowD-current*: *web-flow* Γ *f* $\Longrightarrow$ *current* Γ *f*
$\langle proof \rangle$

lemma *web-flowD-KIR*: $[\![$ *web-flow* Γ *f*; $x \notin A\ \Gamma$; $x \notin B\ \Gamma$ $]\!]$ $\Longrightarrow$ *KIR f x*
$\langle proof \rangle$

6.1 Saturated and terminal vertices

inductive-set *SAT* :: (*'v*, *'more*) *web-scheme* $\Rightarrow$ *'v current* $\Rightarrow$ *'v set*
for Γ *f*
where
A: $x \in A\ \Gamma \Longrightarrow x \in SAT\ \Gamma\ f$
| *IN*: *d-IN f x* $\geq$ *weight* Γ *x* $\Longrightarrow$ $x \in SAT\ \Gamma\ f$
— We use $\geq$ *weight* such that *SAT* is monotone w.r.t. increasing currents

lemma *SAT-0* [*simp*]: *SAT* Γ *zero-current* = $A\ \Gamma \cup \{x.\ weight\ \Gamma\ x \leq 0\}$
$\langle proof \rangle$

lemma *SAT-mono*:
assumes $\bigwedge e.\ f\ e \leq g\ e$
shows $SAT\ \Gamma\ f \subseteq SAT\ \Gamma\ g$
$\langle proof \rangle$

lemma *SAT-Sup-upper*: $f \in Y \Longrightarrow SAT\ \Gamma\ f \subseteq SAT\ \Gamma\ (Sup\ Y)$
$\langle proof \rangle$

lemma *currentD-SAT*:
assumes *current* Γ *f*
shows $x \in SAT\ \Gamma\ f \longleftrightarrow x \in A\ \Gamma \lor$ *d-IN f x* = *weight* Γ *x*
$\langle proof \rangle$

abbreviation *terminal* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *'v current* $\Rightarrow$ *'v set* (*TER*1)
where *terminal* Γ $f \equiv SAT\ \Gamma\ f \cap SINK\ f$

6.2 Separation

inductive *separating-gen* :: $('v, 'more)$ *graph-scheme* $\Rightarrow$ *'v set* $\Rightarrow$ *'v set* $\Rightarrow$ *'v set* $\Rightarrow$ *bool*
for $G\ A\ B\ S$
where *separating*:
$(\bigwedge x\ y\ p.\ [\![\ x \in A;\ y \in B;\ path\ G\ x\ p\ y\ ]\!] \Longrightarrow (\exists z \in set\ p.\ z \in S) \vee x \in S)$
$\Longrightarrow separating\text{-}gen\ G\ A\ B\ S$

abbreviation *separating* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *'v set* $\Rightarrow$ *bool*
where *separating* $\Gamma \equiv separating\text{-}gen\ \Gamma\ (A\ \Gamma)\ (B\ \Gamma)$

abbreviation *separating-network* :: $('v, 'more)$ *network-scheme* $\Rightarrow$ *'v set* $\Rightarrow$ *bool*
where *separating-network* $\Delta \equiv separating\text{-}gen\ \Delta\ \{source\ \Delta\}\ \{sink\ \Delta\}$

lemma *separating-networkI* [*intro?*]:
$(\bigwedge p.\ path\ \Delta\ (source\ \Delta)\ p\ (sink\ \Delta) \Longrightarrow (\exists z \in set\ p.\ z \in S) \vee source\ \Delta \in S)$
$\Longrightarrow separating\text{-}network\ \Delta\ S$
$\langle proof \rangle$

lemma *separatingD*:
$\bigwedge A\ B.\ [\![\ separating\text{-}gen\ G\ A\ B\ S;\ path\ G\ x\ p\ y;\ x \in A;\ y \in B\ ]\!] \Longrightarrow (\exists z \in set\ p.\ z \in S) \vee x \in S$
$\langle proof \rangle$

lemma *separating-left* [*simp*]: $\bigwedge A\ B.\ A \subseteq A' \Longrightarrow separating\text{-}gen\ \Gamma\ A\ B\ A'$
$\langle proof \rangle$

lemma *separating-weakening*:
$\bigwedge A\ B.\ [\![\ separating\text{-}gen\ G\ A\ B\ S;\ S \subseteq S'\ ]\!] \Longrightarrow separating\text{-}gen\ G\ A\ B\ S'$
$\langle proof \rangle$

definition *essential* :: $('v, 'more)$ *graph-scheme* $\Rightarrow$ *'v set* $\Rightarrow$ *'v set* $\Rightarrow$ *'v* $\Rightarrow$ *bool*
where — Should we allow only simple paths here?
$\bigwedge B.\ essential\ G\ B\ S\ x \longleftrightarrow (\exists p.\ \exists y{\in}B.\ path\ G\ x\ p\ y \wedge (x \neq y \longrightarrow (\forall z{\in}set\ p.\ z = x \vee z \notin S)))$

abbreviation *essential-web* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *'v set* $\Rightarrow$ *'v set* ($\mathcal{E}$1)
where *essential-web* $\Gamma\ S \equiv \{x{\in}S.\ essential\ \Gamma\ (B\ \Gamma)\ S\ x\}$

lemma *essential-weight-update* [*simp*]:
essential (*weight-update* f G) = *essential* G
$\langle proof \rangle$

lemma *not-essentialD*:

$\bigwedge B.\ [\![\ \neg\ essential\ G\ B\ S\ x;\ path\ G\ x\ p\ y;\ y \in B\ ]\!] \Longrightarrow x \neq y \wedge (\exists z{\in}set\ p.\ z \neq x \wedge z \in S)$
⟨*proof*⟩

lemma *essentialE* [*elim?*, *consumes 1*, *case-names essential*, *cases pred*: *essential*]:
$\bigwedge B.\ [\![\ essential\ G\ B\ S\ x;\ \bigwedge p\ y.\ [\![\ path\ G\ x\ p\ y;\ y \in B;\ \bigwedge z.\ [\![\ x \neq y;\ z \in set\ p\ ]\!] \Longrightarrow z = x \vee z \notin S\ ]\!] \Longrightarrow thesis\ ]\!] \Longrightarrow thesis$
⟨*proof*⟩

lemma *essentialI* [*intro?*]:
$\bigwedge B.\ [\![\ path\ G\ x\ p\ y;\ y \in B;\ \bigwedge z.\ [\![\ x \neq y;\ z \in set\ p\ ]\!] \Longrightarrow z = x \vee z \notin S\ ]\!] \Longrightarrow essential\ G\ B\ S\ x$
⟨*proof*⟩

lemma *essential-vertex*: $\bigwedge B.\ [\![\ essential\ G\ B\ S\ x;\ x \notin B\ ]\!] \Longrightarrow vertex\ G\ x$
⟨*proof*⟩

lemma *essential-BI*: $\bigwedge B.\ x \in B \Longrightarrow essential\ G\ B\ S\ x$
⟨*proof*⟩

lemma $\mathcal{E}$*-E* [*elim?*, *consumes 1*, *case-names* $\mathcal{E}$, *cases set*: *essential-web*]:
fixes Γ (**structure**)
assumes $x \in \mathcal{E}\ S$
obtains $p\ y$ **where** $path\ \Gamma\ x\ p\ y\ y \in B\ \Gamma\ \bigwedge z.\ [\![\ x \neq y;\ z \in set\ p\ ]\!] \Longrightarrow z = x \vee z \notin S$
⟨*proof*⟩

lemma *essential-mono*: $\bigwedge B.\ [\![\ essential\ G\ B\ S\ x;\ S' \subseteq S\ ]\!] \Longrightarrow essential\ G\ B\ S'\ x$
⟨*proof*⟩

lemma *separating-essential*: — Lem. 3.4 (cf. Lem. 2.14 in [5])
fixes $G\ A\ B\ S$
assumes *separating-gen* $G\ A\ B\ S$
shows *separating-gen* $G\ A\ B\ \{x{\in}S.\ essential\ G\ B\ S\ x\}$ (**is** *separating-gen - - - ?E*)
⟨*proof*⟩

definition *roofed-gen* :: ('*v*, '*more*) *graph-scheme* ⇒ '*v set* ⇒ '*v set* ⇒ '*v set*
where *roofed-def*: $\bigwedge B.\ roofed\text{-}gen\ G\ B\ S = \{x.\ \forall p.\ \forall y{\in}B.\ path\ G\ x\ p\ y \longrightarrow (\exists z{\in}set\ p.\ z \in S) \vee x \in S\}$

abbreviation *roofed* :: ('*v*, '*more*) *web-scheme* ⇒ '*v set* ⇒ '*v set* (RF_1)
where *roofed* $\Gamma \equiv$ *roofed-gen* Γ ($B\ \Gamma$)

abbreviation *roofed-network* :: ('*v*, '*more*) *network-scheme* ⇒ '*v set* ⇒ '*v set* ($RF^N{}_1$)
where *roofed-network* $\Delta \equiv$ *roofed-gen* Δ $\{sink\ \Delta\}$

lemma *roofedI* [*intro?*]:

$\bigwedge B.$ ($\bigwedge p\ y.$ $[\![$ *path* $G\ x\ p\ y$; $y \in B$ $]\!] \Longrightarrow (\exists z{\in}set\ p.\ z \in S) \vee x \in S) \Longrightarrow x \in$ *roofed-gen* $G\ B\ S$
⟨*proof*⟩

lemma *not-roofedE*: **fixes** B
assumes $x \notin$ *roofed-gen* $G\ B\ S$
obtains $p\ y$ **where** *path* $G\ x\ p\ y$ $y \in B$ $\bigwedge z.\ z \in set\ (x\ \#\ p) \Longrightarrow z \notin S$
⟨*proof*⟩

lemma *roofed-greater*: $\bigwedge B.\ S \subseteq$ *roofed-gen* $G\ B\ S$
⟨*proof*⟩

lemma *roofed-greaterI*: $\bigwedge B.\ x \in S \Longrightarrow x \in$ *roofed-gen* $G\ B\ S$
⟨*proof*⟩

lemma *roofed-mono*: $\bigwedge B.\ S \subseteq S' \Longrightarrow$ *roofed-gen* $G\ B\ S \subseteq$ *roofed-gen* $G\ B\ S'$
⟨*proof*⟩

lemma *in-roofed-mono*: $\bigwedge B.$ $[\![$ $x \in$ *roofed-gen* $G\ B\ S$; $S \subseteq S'$ $]\!] \Longrightarrow x \in$ *roofed-gen* $G\ B\ S'$
⟨*proof*⟩

lemma *roofedD*: $\bigwedge B.$ $[\![$ $x \in$ *roofed-gen* $G\ B\ S$; *path* $G\ x\ p\ y$; $y \in B$ $]\!] \Longrightarrow (\exists z{\in}set\ p.\ z \in S) \vee x \in S$
⟨*proof*⟩

lemma *separating-RF-A*:
fixes $A\ B$
assumes *separating-gen* $G\ A\ B\ X$
shows $A \subseteq$ *roofed-gen* $G\ B\ X$
⟨*proof*⟩

lemma *roofed-idem*: **fixes** B **shows** *roofed-gen* $G\ B$ (*roofed-gen* $G\ B\ S$) = *roofed-gen* $G\ B\ S$
⟨*proof*⟩

lemma *in-roofed-mono'*: $\bigwedge B.$ $[\![$ $x \in$ *roofed-gen* $G\ B\ S$; $S \subseteq$ *roofed-gen* $G\ B\ S'$ $]\!] \Longrightarrow x \in$ *roofed-gen* $G\ B\ S'$
⟨*proof*⟩

lemma *roofed-mono'*: $\bigwedge B.\ S \subseteq$ *roofed-gen* $G\ B\ S' \Longrightarrow$ *roofed-gen* $G\ B\ S \subseteq$ *roofed-gen* $G\ B\ S'$
⟨*proof*⟩

lemma *roofed-idem-Un1*: **fixes** B **shows** *roofed-gen* $G\ B$ (*roofed-gen* $G\ B\ S \cup T$) = *roofed-gen* $G\ B\ (S \cup T)$
⟨*proof*⟩

lemma *roofed-UN*: **fixes** $A\ B$

shows *roofed-gen* G B ($\bigcup i{\in}A.$ *roofed-gen* G B $(X\ i)$) $=$ *roofed-gen* G B ($\bigcup i{\in}A.$ $X\ i$) (**is** *?lhs* $=$ *?rhs*)
⟨*proof*⟩

lemma *RF-essential*: **fixes** Γ (**structure**) **shows** $RF\ (\mathcal{E}\ S) = RF\ S$
⟨*proof*⟩

lemma *essentialE-RF*:
fixes Γ (**structure**) **and** B
assumes *essential* $\Gamma\ B\ S\ x$
obtains $p\ y$ **where** *path* $\Gamma\ x\ p\ y$ $y \in B$ *distinct* $(x\ \#\ p)$ $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin$ *roofed-gen* $\Gamma\ B\ S$
⟨*proof*⟩

lemma $\mathcal{E}$*-E-RF*:
fixes Γ (**structure**)
assumes $x \in \mathcal{E}\ S$
obtains $p\ y$ **where** *path* $\Gamma\ x\ p\ y$ $y \in B\ \Gamma$ *distinct* $(x\ \#\ p)$ $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF\ S$
⟨*proof*⟩

lemma *in-roofed-essentialD*:
fixes Γ (**structure**)
assumes *RF*: $x \in RF\ S$
and *ess*: *essential* $\Gamma\ (B\ \Gamma)\ S\ x$
shows $x \in S$
⟨*proof*⟩

lemma *separating-RF*: **fixes** Γ (**structure**) **shows** *separating* $\Gamma\ (RF\ S) \longleftrightarrow$ *separating* $\Gamma\ S$
⟨*proof*⟩

definition *roofed-circ* :: $('v,\ 'more)$ *web-scheme* $\Rightarrow$ $'v\ set \Rightarrow\ 'v\ set$ $(RF^{\circ}1)$
where *roofed-circ* $\Gamma\ S =$ *roofed* $\Gamma\ S - \mathcal{E}_{\Gamma}\ S$

lemma *roofed-circI*: **fixes** Γ (**structure**) **shows**
$[\![\ x \in RF\ T;\ x \in T \Longrightarrow \neg$ *essential* $\Gamma\ (B\ \Gamma)\ T\ x\ ]\!] \Longrightarrow x \in RF^{\circ}\ T$
⟨*proof*⟩

lemma *roofed-circE*:
fixes Γ (**structure**)
assumes $x \in RF^{\circ}\ T$
obtains $x \in RF\ T$ $\neg$ *essential* $\Gamma\ (B\ \Gamma)\ T\ x$
⟨*proof*⟩

lemma $\mathcal{E}$-$\mathcal{E}$: **fixes** Γ (**structure**) **shows** $\mathcal{E}\ (\mathcal{E}\ S) = \mathcal{E}\ S$
⟨*proof*⟩

lemma *roofed-circ-essential*: **fixes** Γ (**structure**) **shows** $RF^{\circ}\ (\mathcal{E}\ S) = RF^{\circ}\ S$

$\langle proof \rangle$

lemma *essential-RF*: **fixes** B
shows *essential* G B (*roofed-gen* G B S) = *essential* G B S (**is** *essential* - - *?RF* = -)
$\langle proof \rangle$

lemma $\mathcal{E}$-*RF*: **fixes** Γ (**structure**) **shows** $\mathcal{E}$ (*RF* S) = $\mathcal{E}$ S
$\langle proof \rangle$

lemma *essential*-$\mathcal{E}$: **fixes** Γ (**structure**) **shows** *essential* Γ (B Γ) ($\mathcal{E}$ S) = *essential* Γ (B Γ) S
$\langle proof \rangle$

lemma *RF-in-B*: **fixes** Γ (**structure**) **shows** $x \in B\ \Gamma \Longrightarrow x \in RF\ S \longleftrightarrow x \in S$
$\langle proof \rangle$

lemma *RF-circ-edge-forward*:
fixes Γ (**structure**)
assumes x: $x \in RF^{\circ}\ S$
and *edge*: *edge* Γ x y
shows $y \in RF\ S$
$\langle proof \rangle$

6.3 Waves

inductive *wave* :: ($'v$, $'more$) *web-scheme* $\Rightarrow$ $'v$ *current* $\Rightarrow$ *bool*
for Γ (**structure**) **and** f
where
wave:
$\llbracket$ *separating* Γ (*TER* f);
$\bigwedge x.\ x \notin RF\ (TER\ f) \Longrightarrow d\text{-}OUT\ f\ x = 0$ $\rrbracket$
$\Longrightarrow$ *wave* Γ f

lemma *wave-0* [*simp*]: *wave* Γ *zero-current*
$\langle proof \rangle$

lemma *waveD-separating*: *wave* Γ $f \Longrightarrow$ *separating* Γ (TER_{Γ} f)
$\langle proof \rangle$

lemma *waveD-OUT*: $\llbracket$ *wave* Γ f; $x \notin RF_{\Gamma}\ (TER_{\Gamma}\ f)$ $\rrbracket \Longrightarrow d\text{-}OUT\ f\ x = 0$
$\langle proof \rangle$

lemma *wave-A-in-RF*: **fixes** Γ (**structure**)
shows $\llbracket$ *wave* Γ f; $x \in A\ \Gamma$ $\rrbracket \Longrightarrow x \in RF\ (TER\ f)$
$\langle proof \rangle$

lemma *wave-not-RF-IN-zero*:
fixes Γ (**structure**)

assumes f: *current* Γ f
and w: *wave* Γ f
and x: $x \notin$ *RF* (*TER* f)
shows *d-IN* f $x = 0$
⟨*proof*⟩

lemma *current-Sup*:
fixes Γ (**structure**)
assumes *chain*: *Complete-Partial-Order.chain* ($\leq$) Y
and Y: $Y \neq \{\}$
and *current*: $\bigwedge f.\ f \in Y \Longrightarrow$ *current* Γ f
and *countable* [*simp*]: *countable* (*support-flow* (*Sup* Y))
shows *current* Γ (*Sup* Y)
⟨*proof*⟩

lemma *wave-lub*: — Lemma 4.3
fixes Γ (**structure**)
assumes *chain*: *Complete-Partial-Order.chain* ($\leq$) Y
and Y: $Y \neq \{\}$
and *wave*: $\bigwedge f.\ f \in Y \Longrightarrow$ *wave* Γ f
and *countable* [*simp*]: *countable* (*support-flow* (*Sup* Y))
shows *wave* Γ (*Sup* Y)
⟨*proof*⟩

lemma *ex-maximal-wave*: — Corollary 4.4
fixes Γ (**structure**)
assumes *countable*: *countable* **E**
shows $\exists f.$ *current* Γ f $\wedge$ *wave* Γ f $\wedge$ ($\forall w.$ *current* Γ w $\wedge$ *wave* Γ w $\wedge$ $f \leq w$ $\longrightarrow f = w$)
⟨*proof*⟩

lemma *essential-leI*:
fixes Γ (**structure**)
assumes g: *current* Γ g **and** w: *wave* Γ g
and *le*: $\bigwedge e.\ f\ e \leq g\ e$
and x: $x \in \mathcal{E}$ (*TER* g)
shows *essential* Γ (B Γ) (*TER* f) x
⟨*proof*⟩

lemma *essential-eq-leI*:
fixes Γ (**structure**)
assumes g: *current* Γ g **and** w: *wave* Γ g
and *le*: $\bigwedge e.\ f\ e \leq g\ e$
and *subset*: $\mathcal{E}$ (*TER* g) $\subseteq$ *TER* f
shows $\mathcal{E}$ (*TER* f) $= \mathcal{E}$ (*TER* g)
⟨*proof*⟩

6.4 Hindrances and looseness

inductive *hindrance-by* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *'v current* $\Rightarrow$ *ennreal* $\Rightarrow$ *bool*
for Γ (**structure**) **and** f **and** ε
where
hindrance-by:
$[\![a \in A\ \Gamma;\ a \notin \mathcal{E}\ (TER\ f);\ d\text{-}OUT\ f\ a < weight\ \Gamma\ a;\ \varepsilon < weight\ \Gamma\ a - d\text{-}OUT\ f\ a]\!] \Longrightarrow hindrance\text{-}by\ \Gamma\ f\ \varepsilon$

inductive *hindrance* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *'v current* $\Rightarrow$ *bool*
for Γ (**structure**) **and** f
where
hindrance:
$[\![a \in A\ \Gamma;\ a \notin \mathcal{E}\ (TER\ f);\ d\text{-}OUT\ f\ a < weight\ \Gamma\ a]\!] \Longrightarrow hindrance\ \Gamma\ f$

inductive *hindered* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *bool*
for Γ (**structure**)
where *hindered*: $[\![hindrance\ \Gamma\ f;\ current\ \Gamma\ f;\ wave\ \Gamma\ f]\!] \Longrightarrow hindered\ \Gamma$

inductive *hindered-by* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *ennreal* $\Rightarrow$ *bool*
for Γ (**structure**) **and** ε
where *hindered-by*: $[\![hindrance\text{-}by\ \Gamma\ f\ \varepsilon;\ current\ \Gamma\ f;\ wave\ \Gamma\ f]\!] \Longrightarrow hindered\text{-}by\ \Gamma\ \varepsilon$

lemma *hindrance-into-hindrance-by*:
assumes *hindrance* Γ f
shows $\exists \varepsilon > 0.\ hindrance\text{-}by\ \Gamma\ f\ \varepsilon$
⟨*proof*⟩

lemma *hindrance-by-into-hindrance*: $hindrance\text{-}by\ \Gamma\ f\ \varepsilon \Longrightarrow hindrance\ \Gamma\ f$
⟨*proof*⟩

lemma *hindrance-conv-hindrance-by*: $hindrance\ \Gamma\ f \longleftrightarrow (\exists \varepsilon > 0.\ hindrance\text{-}by\ \Gamma\ f\ \varepsilon)$
⟨*proof*⟩

lemma *hindered-into-hindered-by*: $hindered\ \Gamma \Longrightarrow \exists \varepsilon > 0.\ hindered\text{-}by\ \Gamma\ \varepsilon$
⟨*proof*⟩

lemma *hindered-by-into-hindered*: $hindered\text{-}by\ \Gamma\ \varepsilon \Longrightarrow hindered\ \Gamma$
⟨*proof*⟩

lemma *hindered-conv-hindered-by*: $hindered\ \Gamma \longleftrightarrow (\exists \varepsilon > 0.\ hindered\text{-}by\ \Gamma\ \varepsilon)$
⟨*proof*⟩

inductive *loose* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *bool*
for Γ
where
loose: $[\![\bigwedge f.\ [\![current\ \Gamma\ f;\ wave\ \Gamma\ f]\!] \Longrightarrow f = zero\text{-}current;\ \neg\ hindrance\ \Gamma\ zero\text{-}current]\!]$

$\Longrightarrow$ *loose* Γ

lemma *looseD-hindrance*: *loose* $\Gamma \Longrightarrow \neg$ *hindrance* Γ *zero-current*
⟨*proof*⟩

lemma *looseD-wave*:
$[\![$ *loose* Γ; *current* Γ f; *wave* Γ f $]\!] \Longrightarrow f =$ *zero-current*
⟨*proof*⟩

lemma *loose-unhindered*:
fixes Γ (**structure**)
assumes *loose* Γ
shows $\neg$ *hindered* Γ
⟨*proof*⟩

context
fixes Γ Γ' :: ($'v$, $'more$) *web-scheme*
assumes [*simp*]: *edge* Γ = *edge* Γ' A $\Gamma = A$ Γ' B $\Gamma = B$ Γ'
and *weight-eq*: $\bigwedge x.\ x \notin A\ \Gamma' \Longrightarrow$ *weight* Γ x = *weight* Γ' x
and *weight-le*: $\bigwedge a.\ a \in A\ \Gamma' \Longrightarrow$ *weight* Γ $a \geq$ *weight* Γ' a
begin

private lemma *essential-eq*: *essential* Γ = *essential* Γ'
⟨*proof*⟩ **lemma** *TER-eq*: $TER_\Gamma\ f = TER_{\Gamma'}\ f$
⟨*proof*⟩ **lemma** *separating-eq*: *separating-gen* Γ = *separating-gen* Γ'
⟨*proof*⟩ **lemma** *roofed-eq*: $\bigwedge B$. *roofed-gen* Γ B S = *roofed-gen* Γ' B S
⟨*proof*⟩

lemma *wave-eq-web*: — Observation 4.6
wave Γ $f \longleftrightarrow$ *wave* Γ' f
⟨*proof*⟩

lemma *current-mono-web*: *current* Γ' $f \Longrightarrow$ *current* Γ f
⟨*proof*⟩

lemma *hindrance-mono-web*: *hindrance* Γ' $f \Longrightarrow$ *hindrance* Γ f
⟨*proof*⟩

lemma *hindered-mono-web*: *hindered* $\Gamma' \Longrightarrow$ *hindered* Γ
⟨*proof*⟩

end

6.5 Linkage

The following definition of orthogonality is stronger than the original definition 3.5 in [2] in that the outflow from any A-vertices in the set must saturate the vertex; $S \subseteq SAT\ \Gamma\ f$ is not enough.

With the original definition of orthogonal current, the reduction from networks to webs fails because the induced flow need not saturate edges going out of the source. Consider the network with three nodes s, x, and t and edges (s, x) and (x, t) with capacity 1. Then, the corresponding web has the vertices (s, x) and (x, t) and one edge from (s, x) to (x, t). Clearly, the zero current *zero-current* is a web-flow and *TER zero-current* $= \{(s, x)\}$, which is essential. Moreover, *zero-current* and $\{(s, x)\}$ are orthogonal because *zero-current* trivially saturates (s, x) as this is a vertex in A.

inductive *orthogonal-current* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ $'v$ *current* $\Rightarrow$ $'v$ *set* $\Rightarrow$ *bool*
  for Γ (**structure**) **and** *f S*
where *orthogonal-current*:
  $[\![\ \bigwedge x.\ [\![\ x \in S;\ x \notin A\ \Gamma\ ]\!] \Longrightarrow$ *weight* $\Gamma\ x \leq$ *d-IN f x*;
    $\bigwedge x.\ [\![\ x \in S;\ x \in A\ \Gamma;\ x \notin B\ \Gamma\ ]\!] \Longrightarrow$ *d-OUT f x* $=$ *weight* $\Gamma\ x$;
    $\bigwedge u\ v.\ [\![\ v \in RF\ S;\ u \notin RF^{\circ}\ S\ ]\!] \Longrightarrow f\ (u, v) = 0\ ]\!]$
  $\Longrightarrow$ *orthogonal-current* $\Gamma\ f\ S$

lemma *orthogonal-currentD-SAT*: $[\![$ *orthogonal-current* $\Gamma\ f\ S;\ x \in S\ ]\!] \Longrightarrow x \in$ *SAT* $\Gamma\ f$
⟨*proof*⟩

lemma *orthogonal-currentD-A*: $[\![$ *orthogonal-current* $\Gamma\ f\ S;\ x \in S;\ x \in A\ \Gamma;\ x \notin B\ \Gamma\ ]\!] \Longrightarrow$ *d-OUT f x* $=$ *weight* $\Gamma\ x$
⟨*proof*⟩

lemma *orthogonal-currentD-in*: $[\![$ *orthogonal-current* $\Gamma\ f\ S;\ v \in RF_{\Gamma}\ S;\ u \notin RF^{\circ}{}_{\Gamma}\ S\ ]\!] \Longrightarrow f\ (u, v) = 0$
⟨*proof*⟩

inductive *linkage* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ $'v$ *current* $\Rightarrow$ *bool*
  for $\Gamma\ f$
where — Omit the condition *web-flow*
  linkage: $(\bigwedge x.\ x \in A\ \Gamma \Longrightarrow$ *d-OUT f x* $=$ *weight* $\Gamma\ x) \Longrightarrow$ *linkage* $\Gamma\ f$

lemma *linkageD*: $[\![$ *linkage* $\Gamma\ f;\ x \in A\ \Gamma\ ]\!] \Longrightarrow$ *d-OUT f x* $=$ *weight* $\Gamma\ x$
⟨*proof*⟩

abbreviation *linkable* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *bool*
where *linkable* $\Gamma \equiv \exists f.$ *web-flow* $\Gamma\ f \wedge$ *linkage* $\Gamma\ f$

6.6 Trimming

context
  fixes Γ :: $('v, 'more)$ *web-scheme* (**structure**)
  and f :: $'v$ *current*
begin

inductive *trimming* :: $'v$ *current* $\Rightarrow$ *bool*
  for g

where
trimming:
— omits the condition that f is a wave
$\llbracket$ *current* Γ g; *wave* Γ g; $g \le f$; $\bigwedge x$. $\llbracket$ $x \in RF^\circ$ (*TER* f); $x \notin A\ \Gamma$ $\rrbracket \Longrightarrow$ *KIR* g x; $\mathcal{E}$ (*TER* g) $-$ $A\ \Gamma = \mathcal{E}$ (*TER* f) $-$ $A\ \Gamma$ $\rrbracket$
$\Longrightarrow$ *trimming* g

lemma assumes *trimming* g
shows *trimmingD-current*: *current* Γ g
and *trimmingD-wave*: *wave* Γ g
and *trimmingD-le*: $\bigwedge e.\ g\ e \le f\ e$
and *trimmingD-KIR*: $\llbracket$ $x \in RF^\circ$ (*TER* f); $x \notin A\ \Gamma$ $\rrbracket \Longrightarrow$ *KIR* g x
and *trimmingD-$\mathcal{E}$*: $\mathcal{E}$ (*TER* g) $-$ $A\ \Gamma = \mathcal{E}$ (*TER* f) $-$ $A\ \Gamma$
$\langle$*proof*$\rangle$

lemma *ex-trimming*: — Lemma 4.8
assumes f: *current* Γ f
and w: *wave* Γ f
and *countable*: *countable* **E**
and *weight-finite*: $\bigwedge x$. *weight* Γ $x \neq \top$
shows $\exists\, g$. *trimming* g
$\langle$*proof*$\rangle$

end

lemma *trimming-$\mathcal{E}$*:
fixes Γ (**structure**)
assumes w: *wave* Γ f **and** *trimming*: *trimming* Γ f g
shows $\mathcal{E}$ (*TER* f) $= \mathcal{E}$ (*TER* g)
$\langle$*proof*$\rangle$

6.7 Composition of waves via quotients

definition *quotient-web* :: ($'v$, $'more$) *web-scheme* $\Rightarrow$ $'v$ *current* $\Rightarrow$ ($'v$, $'more$) *web-scheme*
where — Modifications to original Definition 4.9: No incoming edges to nodes in A, $B\ \Gamma - A\ \Gamma$ is not part of A such that A contains only vertices is disjoint from B. The weight of vertices in B saturated by f is therefore set to *0*.
quotient-web Γ f $=$
$(\!|$*edge* $= \lambda x\ y$. *edge* Γ x y $\wedge$ $x \notin$ *roofed-circ* Γ (*TER*$_\Gamma$ f) $\wedge$ $y \notin$ *roofed* Γ (*TER*$_\Gamma$ f),
weight $= \lambda x$. *if* $x \in RF^\circ{}_\Gamma$ (*TER*$_\Gamma$ f) $\vee$ $x \in$ *TER*$_\Gamma$ $f \cap B\ \Gamma$ *then 0 else weight* Γ x,
$A = \mathcal{E}_\Gamma$ (*TER*$_\Gamma$ f) $-$ ($B\ \Gamma - A\ \Gamma$),
$B = B\ \Gamma$,
$\ldots =$ *web.more* $\Gamma|\!)$

lemma *quotient-web-sel* [*simp*]:
fixes Γ (**structure**) **shows**

$edge\ (quotient\text{-}web\ \Gamma\ f)\ x\ y \longleftrightarrow edge\ \Gamma\ x\ y \land x \notin RF^{\circ}\ (TER\ f) \land y \notin RF\ (TER\ f)$
$weight\ (quotient\text{-}web\ \Gamma\ f)\ x = (if\ x \in RF^{\circ}\ (TER\ f) \lor x \in TER_{\Gamma}\ f \cap B\ \Gamma\ then\ 0\ else\ weight\ \Gamma\ x)$
$A\ (quotient\text{-}web\ \Gamma\ f) = \mathcal{E}\ (TER\ f) - (B\ \Gamma - A\ \Gamma)$
$B\ (quotient\text{-}web\ \Gamma\ f) = B\ \Gamma$
$web.more\ (quotient\text{-}web\ \Gamma\ f) = web.more\ \Gamma$
$\langle proof \rangle$

lemma *vertex-quotient-webD*: **fixes** Γ (**structure**) **shows**
$vertex\ (quotient\text{-}web\ \Gamma\ f)\ x \Longrightarrow vertex\ \Gamma\ x \land x \notin RF^{\circ}\ (TER\ f)$
$\langle proof \rangle$

lemma *path-quotient-web*:
fixes Γ (**structure**)
assumes $path\ \Gamma\ x\ p\ y$
and $x \notin RF^{\circ}\ (TER\ f)$
and $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF\ (TER\ f)$
shows $path\ (quotient\text{-}web\ \Gamma\ f)\ x\ p\ y$
$\langle proof \rangle$

definition *restrict-current* :: $('v, 'more)\ web\text{-}scheme \Rightarrow 'v\ current \Rightarrow 'v\ current \Rightarrow 'v\ current$
where $restrict\text{-}current\ \Gamma\ f\ g = (\lambda(x, y).\ g\ (x, y) * indicator\ (-\ RF^{\circ}{}_{\Gamma}\ (TER_{\Gamma}\ f))\ x * indicator\ (-\ RF_{\Gamma}\ (TER_{\Gamma}\ f))\ y)$

abbreviation *restrict-curr* :: $'v\ current \Rightarrow ('v, 'more)\ web\text{-}scheme \Rightarrow 'v\ current \Rightarrow 'v\ current$ (- ↿ - '/ - [100, 0, 100] 100)
where $restrict\text{-}curr\ g\ \Gamma\ f \equiv restrict\text{-}current\ \Gamma\ f\ g$

lemma *restrict-current-simps* [*simp*]: **fixes** Γ (**structure**) **shows**
$(g \upharpoonleft \Gamma\ /\ f)\ (x, y) = (g\ (x, y) * indicator\ (-\ RF^{\circ}\ (TER\ f))\ x * indicator\ (-\ RF\ (TER\ f))\ y)$
$\langle proof \rangle$

lemma *d-OUT-restrict-current-outside*: **fixes** Γ (**structure**) **shows**
$x \in RF^{\circ}\ (TER\ f) \Longrightarrow d\text{-}OUT\ (g \upharpoonleft \Gamma\ /\ f)\ x = 0$
$\langle proof \rangle$

lemma *d-IN-restrict-current-outside*: **fixes** Γ (**structure**) **shows**
$x \in RF\ (TER\ f) \Longrightarrow d\text{-}IN\ (g \upharpoonleft \Gamma\ /\ f)\ x = 0$
$\langle proof \rangle$

lemma *restrict-current-le*: $(g \upharpoonleft \Gamma\ /\ f)\ e \leq g\ e$
$\langle proof \rangle$

lemma *d-OUT-restrict-current-le*: $d\text{-}OUT\ (g \upharpoonleft \Gamma\ /\ f)\ x \leq d\text{-}OUT\ g\ x$
$\langle proof \rangle$

lemma *d-IN-restrict-current-le*: *d-IN* $(g \upharpoonright \Gamma \;/\; f)$ $x \leq$ *d-IN* g x
⟨*proof*⟩

lemma *restrict-current-IN-not-RF*:
fixes Γ (**structure**)
assumes g: *current* Γ g
and x: $x \notin$ *RF* (*TER* f)
shows *d-IN* $(g \upharpoonright \Gamma \;/\; f)$ x = *d-IN* g x
⟨*proof*⟩

lemma *restrict-current-IN-A*:
$a \in A$ (*quotient-web* Γ f) $\Longrightarrow$ *d-IN* $(g \upharpoonright \Gamma \;/\; f)$ a = *0*
⟨*proof*⟩

lemma *restrict-current-nonneg*: *0* $\leq g$ $e \Longrightarrow$ *0* $\leq (g \upharpoonright \Gamma \;/\; f)$ e
⟨*proof*⟩

lemma *in-SINK-restrict-current*: $x \in$ *SINK* $g \Longrightarrow x \in$ *SINK* $(g \upharpoonright \Gamma \;/\; f)$
⟨*proof*⟩

lemma *SAT-restrict-current*:
fixes Γ (**structure**)
assumes f: *current* Γ f
and g: *current* Γ g
shows *SAT* (*quotient-web* Γ f) $(g \upharpoonright \Gamma \;/\; f)$ = *RF* (*TER* f) $\cup$ (*SAT* Γ g $-$ A Γ)
(**is** *SAT* *?Γ* *?g* = *?rhs*)
⟨*proof*⟩

lemma *current-restrict-current*:
fixes Γ (**structure**)
assumes w: *wave* Γ f
and g: *current* Γ g
shows *current* (*quotient-web* Γ f) $(g \upharpoonright \Gamma \;/\; f)$ (**is** *current* *?Γ* *?g*)
⟨*proof*⟩

lemma *TER-restrict-current*:
fixes Γ (**structure**)
assumes f: *current* Γ f
and w: *wave* Γ f
and g: *current* Γ g
shows *TER* $g \subseteq$ $TER_{quotient\text{-}web\ \Gamma\ f}$ $(g \upharpoonright \Gamma \;/\; f)$ (**is** - $\subseteq$ *?TER* **is** - $\subseteq$ $TER_{?\Gamma}$
?g)
⟨*proof*⟩

lemma *wave-restrict-current*:
fixes Γ (**structure**)
assumes f: *current* Γ f
and w: *wave* Γ f
and g: *current* Γ g

and w': *wave* Γ *g*
shows *wave* (*quotient-web* Γ *f*) ($g \upharpoonright \Gamma$ / *f*) (**is** *wave* ?Γ ?*g*)
⟨*proof*⟩

definition *plus-current* :: $'v$ *current* $\Rightarrow$ $'v$ *current* $\Rightarrow$ $'v$ *current*
where *plus-current f g* = (λe. *f e* + *g e*)

lemma *plus-current-simps* [*simp*]: *plus-current f g e* = *f e* + *g e*
⟨*proof*⟩

lemma *plus-zero-current* [*simp*]: *plus-current f zero-current* = *f*
⟨*proof*⟩

lemma *support-flow-plus-current*: *support-flow* (*plus-current f g*) $\subseteq$ *support-flow f* $\cup$ *support-flow g*
⟨*proof*⟩

context
fixes Γ :: ($'v$, $'more$) *web-scheme* (**structure**) **and** *f g*
assumes *f*: *current* Γ *f*
and *w*: *wave* Γ *f*
and *g*: *current* (*quotient-web* Γ *f*) *g*
begin

lemma *OUT-plus-current*: *d-OUT* (*plus-current f g*) *x* = (*if* $x \in RF^{\circ}$ (*TER f*) *then d-OUT f x else d-OUT g x*) (**is** *d-OUT ?g* - = -)
⟨*proof*⟩

lemma *IN-plus-current*: *d-IN* (*plus-current f g*) *x* = (*if* $x \in RF$ (*TER f*) *then d-IN f x else d-IN g x*) (**is** *d-IN ?g* - = -)
⟨*proof*⟩

lemma *in-TER-plus-current*:
assumes *RF*: $x \notin RF^{\circ}$ (*TER f*)
and *x*: $x \in TER_{quotient\text{-}web\ \Gamma\ f}\ g$ (**is** - $\in$ *?TER* -)
shows $x \in$ *TER* (*plus-current f g*) (**is** - $\in$ *TER ?g*)
⟨*proof*⟩

lemma *current-plus-current*: *current* Γ (*plus-current f g*) (**is** *current* - *?g*)
⟨*proof*⟩

context
assumes w': *wave* (*quotient-web* Γ *f*) *g*
begin

lemma *separating-TER-plus-current*:
assumes *x*: $x \in RF$ (*TER f*) **and** *y*: $y \in B\ \Gamma$ **and** *p*: *path* Γ *x p y*
shows ($\exists z \in set\ p$. $z \in$ *TER* (*plus-current f g*)) $\vee$ $x \in$ *TER* (*plus-current f g*) (**is** - $\vee$ - $\in$ *TER ?g*)

$\langle proof \rangle$

lemma *wave-plus-current*: *wave* Γ (*plus-current f g*) (**is** *wave - ?g*)
$\langle proof \rangle$

end

end

lemma *loose-quotient-web*:
fixes Γ :: ('*v*, '*more*) *web-scheme* (**structure**)
assumes *weight-finite*: $\bigwedge x.$ *weight* Γ *x* ≠ ⊤
and *f*: *current* Γ *f*
and *w*: *wave* Γ *f*
and *maximal*: $\bigwedge w.$ ⟦ *current* Γ *w*; *wave* Γ *w*; *f* ≤ *w* ⟧ ⟹ *f* = *w*
shows *loose* (*quotient-web* Γ *f*) (**is** *loose ?*Γ)
$\langle proof \rangle$

lemma *quotient-web-trimming*:
fixes Γ (**structure**)
assumes *w*: *wave* Γ *f*
and *trimming*: *trimming* Γ *f g*
shows *quotient-web* Γ *f* = *quotient-web* Γ *g* (**is** *?lhs* = *?rhs*)
$\langle proof \rangle$

6.8 Well-formed webs

locale *web* =
fixes Γ :: ('*v*, '*more*) *web-scheme* (**structure**)
assumes *A-in*: *x* ∈ *A* Γ ⟹ ¬ *edge* Γ *y x*
and *B-out*: *x* ∈ *B* Γ ⟹ ¬ *edge* Γ *x y*
and *A-vertex*: *A* Γ ⊆ **V**
and *disjoint*: *A* Γ ∩ *B* Γ = {}
and *no-loop*: $\bigwedge x.$ ¬ *edge* Γ *x x*
and *weight-outside*: $\bigwedge x.$ *x* ∉ **V** ⟹ *weight* Γ *x* = *0*
and *weight-finite* [*simp*]: $\bigwedge x.$ *weight* Γ *x* ≠ ⊤
begin

lemma *web-weight-update*:
assumes $\bigwedge x.$ ¬ *vertex* Γ *x* ⟹ *w x* = *0*
and $\bigwedge x.$ *w x* ≠ ⊤
shows *web* (Γ⦇*weight* := *w*⦈)
$\langle proof \rangle$

lemma *currentI* [*intro?*]:
assumes $\bigwedge x.$ *d-OUT f x* ≤ *weight* Γ *x*
and $\bigwedge x.$ *d-IN f x* ≤ *weight* Γ *x*
and *OUT-IN*: $\bigwedge x.$ ⟦ *x* ∉ *A* Γ; *x* ∉ *B* Γ ⟧ ⟹ *d-OUT f x* ≤ *d-IN f x*
and *outside*: $\bigwedge e.$ *e* ∉ **E** ⟹ *f e* = *0*

shows *current* Γ *f*
⟨*proof*⟩

lemma *currentD-finite-IN*:
assumes *f*: *current* Γ *f*
shows *d-IN f x* $\neq \top$
⟨*proof*⟩

lemma *currentD-finite-OUT*:
assumes *f*: *current* Γ *f*
shows *d-OUT f x* $\neq \top$
⟨*proof*⟩

lemma *currentD-finite*:
assumes *f*: *current* Γ *f*
shows *f e* $\neq \top$
⟨*proof*⟩

lemma *web-quotient-web*: *web* (*quotient-web* Γ *f*) (**is** *web* ?Γ)
⟨*proof*⟩

end

locale *countable-web* = *web* Γ
for Γ :: ('*v*, '*more*) *web-scheme* (**structure**)
+
assumes *countable* [*simp*]: *countable* **E**
begin

lemma *countable-V* [*simp*]: *countable* **V**
⟨*proof*⟩

lemma *countable-web-quotient-web*: *countable-web* (*quotient-web* Γ *f*) (**is** *countable-web* ?Γ)
⟨*proof*⟩

end

6.9 Subtraction of a wave

definition *minus-web* :: ('*v*, '*more*) *web-scheme* ⇒ '*v current* ⇒ ('*v*, '*more*) *web-scheme* (**infixl** ⊖ *65*) — Definition 6.6
where $\Gamma \ominus f = \Gamma$⦇*weight* := $\lambda x.$ *if* $x \in A\ \Gamma$ *then weight* $\Gamma\ x$ − *d-OUT f x else weight* $\Gamma\ x$ + *d-OUT f x* − *d-IN f x*⦈

lemma *minus-web-sel* [*simp*]:
edge ($\Gamma \ominus f$) = *edge* Γ
weight ($\Gamma \ominus f$) x = (*if* $x \in A\ \Gamma$ *then weight* $\Gamma\ x$ − *d-OUT f x else weight* $\Gamma\ x$ + *d-OUT f x* − *d-IN f x*)

A $(\Gamma \ominus f) = A\ \Gamma$
B $(\Gamma \ominus f) = B\ \Gamma$
$\mathbf{V}_{\Gamma \ominus f} = \mathbf{V}_{\Gamma}$
$\mathbf{E}_{\Gamma \ominus f} = \mathbf{E}_{\Gamma}$
web.more $(\Gamma \ominus f)$ = *web.more* Γ
⟨*proof*⟩

lemma *vertex-minus-web* [*simp*]: *vertex* $(\Gamma \ominus f)$ = *vertex* Γ
⟨*proof*⟩

lemma *roofed-gen-minus-web* [*simp*]: *roofed-gen* $(\Gamma \ominus f)$ = *roofed-gen* Γ
⟨*proof*⟩

lemma *minus-zero-current* [*simp*]: $\Gamma \ominus$ *zero-current* = Γ
⟨*proof*⟩

lemma (**in** *web*) *web-minus-web*:
assumes *f*: *current* Γ *f*
shows *web* $(\Gamma \ominus f)$
⟨*proof*⟩

6.10 Bipartite webs

locale *countable-bipartite-web* =
fixes Γ :: (*'v*, *'more*) *web-scheme* (**structure**)
assumes *bipartite-V*: $\mathbf{V} \subseteq A\ \Gamma \cup B\ \Gamma$
and *A-vertex*: $A\ \Gamma \subseteq \mathbf{V}$
and *bipartite-E*: *edge* Γ *x y* $\Longrightarrow x \in A\ \Gamma \wedge y \in B\ \Gamma$
and *disjoint*: $A\ \Gamma \cap B\ \Gamma = \{\}$
and *weight-outside*: $\bigwedge x.\ x \notin \mathbf{V} \Longrightarrow$ *weight* Γ *x* = *0*
and *weight-finite* [*simp*]: $\bigwedge x.$ *weight* Γ $x \neq \top$
and *countable-E* [*simp*]: *countable* $\mathbf{E}$
begin

lemma *not-vertex*: $[\![\ x \notin A\ \Gamma;\ x \notin B\ \Gamma\ ]\!] \Longrightarrow \neg$ *vertex* Γ *x*
⟨*proof*⟩

lemma *no-loop*: $\neg$ *edge* Γ *x x*
⟨*proof*⟩

lemma *edge-antiparallel*: *edge* Γ *x y* $\Longrightarrow \neg$ *edge* Γ *y x*
⟨*proof*⟩

lemma *A-in*: $x \in A\ \Gamma \Longrightarrow \neg$ *edge* Γ *y x*
⟨*proof*⟩

lemma *B-out*: $x \in B\ \Gamma \Longrightarrow \neg$ *edge* Γ *x y*
⟨*proof*⟩

sublocale *countable-web* ⟨*proof*⟩

lemma *currentD-OUT'*:
assumes *f*: *current* Γ *f*
and *x*: $x \notin A\ \Gamma$
shows *d-OUT f x* = *0*
⟨*proof*⟩

lemma *currentD-IN'*:
assumes *f*: *current* Γ *f*
and *x*: $x \notin B\ \Gamma$
shows *d-IN f x* = *0*
⟨*proof*⟩

lemma *current-bipartiteI* [*intro?*]:
assumes *OUT*: $\bigwedge x.$ *d-OUT f x* ≤ *weight* Γ *x*
and *IN*: $\bigwedge x.$ *d-IN f x* ≤ *weight* Γ *x*
and *outside*: $\bigwedge e.\ e \notin \mathbf{E} \Longrightarrow f\ e = 0$
shows *current* Γ *f*
⟨*proof*⟩

lemma *wave-bipartiteI* [*intro?*]:
assumes *sep*: *separating* Γ (*TER f*)
and *f*: *current* Γ *f*
shows *wave* Γ *f*
⟨*proof*⟩

lemma *web-flow-iff*: *web-flow* Γ *f* ⟷ *current* Γ *f*
⟨*proof*⟩

end

end

7 Reductions

theory *MFMC-Reduction* **imports**
MFMC-Web
begin

7.1 From a web to a bipartite web

definition *bipartite-web-of* :: (*'v*, *'more*) *web-scheme* ⇒ (*'v* + *'v*, *'more*) *web-scheme*
where
bipartite-web-of Γ =
(|*edge* = λ*uv uv'*. *case* (*uv*, *uv'*) *of* (*Inl u*, *Inr v*) ⇒ *edge* Γ *u v* ∨ *u* = *v* ∧ *u* ∈ *vertices* Γ ∧ $u \notin A$ Γ ∧ $v \notin B$ Γ | - ⇒ *False*,
weight = λ*uv*. *case uv of Inl u* ⇒ *if u* ∈ *B* Γ *then 0 else weight* Γ *u* | *Inr u* ⇒

if $u \in A\ \Gamma$ *then* 0 *else weight* $\Gamma\ u$,
  $A = Inl$ ‘ (*vertices* $\Gamma - B\ \Gamma$),
  $B = Inr$ ‘ ($- A\ \Gamma$),
  $\ldots = web.more\ \Gamma$⦈

lemma *bipartite-web-of-sel* [*simp*]: **fixes** Γ (**structure**) **shows**
  edge (*bipartite-web-of* Γ) (*Inl u*) (*Inr v*) $\longleftrightarrow$ *edge* $\Gamma\ u\ v \vee u = v \wedge u \in \mathbf{V} \wedge u \notin A\ \Gamma \wedge v \notin B\ \Gamma$
  edge (*bipartite-web-of* Γ) *uv* (*Inl u*) $\longleftrightarrow$ *False*
  edge (*bipartite-web-of* Γ) (*Inr v*) *uv* $\longleftrightarrow$ *False*
  weight (*bipartite-web-of* Γ) (*Inl u*) = (*if* $u \in B\ \Gamma$ *then* 0 *else weight* $\Gamma\ u$)
  weight (*bipartite-web-of* Γ) (*Inr v*) = (*if* $v \in A\ \Gamma$ *then* 0 *else weight* $\Gamma\ v$)
  A (*bipartite-web-of* Γ) = *Inl* ‘ ($\mathbf{V} - B\ \Gamma$)
  B (*bipartite-web-of* Γ) = *Inr* ‘ ($- A\ \Gamma$)
⟨*proof*⟩

lemma *edge-bipartite-webI1*: *edge* $\Gamma\ u\ v \Longrightarrow$ *edge* (*bipartite-web-of* Γ) (*Inl u*) (*Inr v*)
⟨*proof*⟩

lemma *edge-bipartite-webI2*:
  $[\![\ u \in \mathbf{V}_\Gamma;\ u \notin A\ \Gamma;\ u \notin B\ \Gamma\ ]\!] \Longrightarrow$ *edge* (*bipartite-web-of* Γ) (*Inl u*) (*Inr u*)
⟨*proof*⟩

lemma *edge-bipartite-webE*:
  fixes Γ (**structure**)
  assumes *edge* (*bipartite-web-of* Γ) *uv uv′*
  obtains *u v* **where** *uv* = *Inl u uv′* = *Inr v edge* $\Gamma\ u\ v$
    | *u* **where** *uv* = *Inl u uv′* = *Inr u* $u \in \mathbf{V}$ $u \notin A\ \Gamma$ $u \notin B\ \Gamma$
⟨*proof*⟩

lemma *E-bipartite-web*:
  fixes Γ (**structure**) **shows**
  $\mathbf{E}_{bipartite\text{-}web\text{-}of\ \Gamma} = (\lambda(x, y).\ (Inl\ x,\ Inr\ y))$ ‘ $\mathbf{E} \cup (\lambda x.\ (Inl\ x,\ Inr\ x))$ ‘ $(\mathbf{V} - A\ \Gamma - B\ \Gamma)$
⟨*proof*⟩

context *web* **begin**

lemma *vertex-bipartite-web* [*simp*]:
  vertex (*bipartite-web-of* Γ) (*Inl x*) $\longleftrightarrow$ *vertex* $\Gamma\ x \wedge x \notin B\ \Gamma$
  vertex (*bipartite-web-of* Γ) (*Inr x*) $\longleftrightarrow$ *vertex* $\Gamma\ x \wedge x \notin A\ \Gamma$
⟨*proof*⟩

definition *separating-of-bipartite* :: ($'v + 'v$) *set* $\Rightarrow$ $'v$ *set*
where
  separating-of-bipartite S =
  (*let A-S* = *Inl* −‘ *S*; *B-S* = *Inr* −‘ *S in* (*A-S* ∩ *B-S*) ∪ ($A\ \Gamma$ ∩ *A-S*) ∪ ($B\ \Gamma$ ∩ *B-S*))

context
  fixes S :: $('v + 'v)$ *set*
  assumes *sep*: *separating* (*bipartite-web-of* Γ) S
begin

Proof of separation follows [1]

lemma *separating-of-bipartite-aux*:
  assumes *p*: *path* Γ x p y **and** *y*: $y \in B\ \Gamma$
  and *x*: $x \in A\ \Gamma \vee$ *Inr* $x \in S$
  shows ($\exists z \in$*set* p. $z \in$ *separating-of-bipartite* S) $\vee$ $x \in$ *separating-of-bipartite* S
⟨*proof*⟩

lemma *separating-of-bipartite*:
  separating Γ (*separating-of-bipartite* S)
⟨*proof*⟩

end

lemma *current-bipartite-web-finite*:
  assumes *f*: *current* (*bipartite-web-of* Γ) f (**is** *current* ?Γ -)
  shows $f\ e \neq \top$
⟨*proof*⟩

definition *current-of-bipartite* :: $('v + 'v)$ *current* $\Rightarrow$ $'v$ *current*
where *current-of-bipartite* f = ($\lambda(x, y)$. f (*Inl* x, *Inr* y) $*$ *indicator* **E** (x, y))

lemma *current-of-bipartite-simps* [*simp*]: *current-of-bipartite* f (x, y) = f (*Inl* x, *Inr* y) $*$ *indicator* **E** (x, y)
⟨*proof*⟩

lemma *d-OUT-current-of-bipartite*:
  assumes *f*: *current* (*bipartite-web-of* Γ) f
  shows *d-OUT* (*current-of-bipartite* f) x = *d-OUT* f (*Inl* x) $-$ f (*Inl* x, *Inr* x)
⟨*proof*⟩

lemma *d-IN-current-of-bipartite*:
  assumes *f*: *current* (*bipartite-web-of* Γ) f
  shows *d-IN* (*current-of-bipartite* f) x = *d-IN* f (*Inr* x) $-$ f (*Inl* x, *Inr* x)
⟨*proof*⟩

lemma *current-current-of-bipartite*: — Lemma 6.3
  assumes *f*: *current* (*bipartite-web-of* Γ) f (**is** *current* ?Γ -)
  and *w*: *wave* (*bipartite-web-of* Γ) f
  shows *current* Γ (*current-of-bipartite* f) (**is** *current* - ?*f*)
⟨*proof*⟩

lemma *TER-current-of-bipartite*: — Lemma 6.3
  assumes *f*: *current* (*bipartite-web-of* Γ) f (**is** *current* ?Γ -)

and *w*: *wave* (*bipartite-web-of* Γ) *f*
shows *TER* (*current-of-bipartite f*) = *separating-of-bipartite* ($TER_{bipartite\text{-}web\text{-}of}$ Γ *f*)
(**is** *TER ?f* = *separating-of-bipartite ?TER*)
⟨*proof*⟩

lemma *wave-current-of-bipartite*: — Lemma 6.3
assumes *f*: *current* (*bipartite-web-of* Γ) *f* (**is** *current ?*Γ -)
and *w*: *wave* (*bipartite-web-of* Γ) *f*
shows *wave* Γ (*current-of-bipartite f*) (**is** *wave* - *?f*)
⟨*proof*⟩

end

context *countable-web* **begin**

lemma *countable-bipartite-web-of*: *countable-bipartite-web* (*bipartite-web-of* Γ) (**is** *countable-bipartite-web ?*Γ)
⟨*proof*⟩

end

context *web* **begin**

lemma *unhindered-bipartite-web-of*:
assumes *loose*: *loose* Γ
shows ¬ *hindered* (*bipartite-web-of* Γ)
⟨*proof*⟩

lemma (**in** −) *divide-less-1-iff-ennreal*: *a* / *b* < (*1*::*ennreal*) ⟷ (*0* < *b* ∧ *a* < *b* ∨ *b* = *0* ∧ *a* = *0* ∨ *b* = *top*)
⟨*proof*⟩

lemma *linkable-bipartite-web-ofD*:
assumes *link*: *linkable* (*bipartite-web-of* Γ) (**is** *linkable ?*Γ)
and *countable*: *countable* **E**
shows *linkable* Γ
⟨*proof*⟩

end

7.2 Extending a wave by a linkage

lemma *linkage-quotient-webD*:
fixes Γ :: (*'v*, *'more*) *web-scheme* (**structure**) **and** *h g*
defines *k* ≡ *plus-current h g*
assumes *f*: *current* Γ *f*
and *w*: *wave* Γ *f*
and *wg*: *web-flow* (*quotient-web* Γ *f*) *g* (**is** *web-flow ?*Γ -)

and *link*: *linkage* (*quotient-web* Γ *f*) *g*
and *trim*: *trimming* Γ *f h*
shows *web-flow* Γ *k*
and *orthogonal-current* Γ *k* ($\mathcal{E}$ (*TER f*))
⟨*proof*⟩

context *countable-web* **begin**

lemma *ex-orthogonal-current'*: — Lemma 4.15
assumes *loose-linkable*: $\bigwedge f$. ⟦ *current* Γ *f*; *wave* Γ *f*; *loose* (*quotient-web* Γ *f*) ⟧ $\Longrightarrow$ *linkable* (*quotient-web* Γ *f*)
shows $\exists f\ S$. *web-flow* Γ *f* $\wedge$ *separating* Γ *S* $\wedge$ *orthogonal-current* Γ *f S*
⟨*proof*⟩

end

7.3 From a network to a web

definition *web-of-network* :: (*'v*, *'more*) *network-scheme* $\Rightarrow$ (*'v edge*, *'more*) *web-scheme*
where
web-of-network Δ =
(|*edge* = $\lambda(x, y)\ (y', z)$. $y' = y$ $\wedge$ *edge* Δ *x y* $\wedge$ *edge* Δ *y z*,
weight = *capacity* Δ,
A = {(*source* Δ, *x*)|*x*. *edge* Δ (*source* Δ) *x*},
B = {(*x*, *sink* Δ)|*x*. *edge* Δ *x* (*sink* Δ)},
... = *network.more* Δ|)

lemma *web-of-network-sel* [*simp*]:
fixes Δ (**structure**) **shows**
edge (*web-of-network* Δ) *e e'* $\longleftrightarrow$ $e \in$ **E** $\wedge$ $e' \in$ **E** $\wedge$ *snd e* = *fst e'*
weight (*web-of-network* Δ) *e* = *capacity* Δ *e*
A (*web-of-network* Δ) = {(*source* Δ, *x*)|*x*. *edge* Δ (*source* Δ) *x*}
B (*web-of-network* Δ) = {(*x*, *sink* Δ)|*x*. *edge* Δ *x* (*sink* Δ)}
⟨*proof*⟩

lemma *vertex-web-of-network* [*simp*]:
vertex (*web-of-network* Δ) (*x*, *y*) $\longleftrightarrow$ *edge* Δ *x y* $\wedge$ ($\exists z$. *edge* Δ *y z* $\vee$ *edge* Δ *z x*)
⟨*proof*⟩

definition *flow-of-current* :: (*'v*, *'more*) *network-scheme* $\Rightarrow$ *'v edge current* $\Rightarrow$ *'v flow*
where *flow-of-current* Δ *f e* = *max* (*d-OUT f e*) (*d-IN f e*)

lemma *flow-flow-of-current*:
fixes Δ (**structure**) **and** Γ
defines [*simp*]: $\Gamma \equiv$ *web-of-network* Δ
assumes *fw*: *web-flow* Γ *f*
shows *flow* Δ (*flow-of-current* Δ *f*) (**is** *flow* - *?f*)

$\langle proof \rangle$

The reduction of Conjecture 1.2 to Conjecture 3.6 is flawed in [2]. Not every essential A-B separating set of vertices in *web-of-network* Δ is an s-t-cut in Δ, as the following counterexample shows.

The network Δ has five nodes s, t, x, y and z and edges (s, x), (x, y), (y, z), (y, t) and (z, t). For *web-of-network* Δ, the set $S = \{(x, y), (y, z)\}$ is essential and A-B separating. ((x, y) is essential due to the path $[(y, z)]$ and (y, z) is essential due to the path $[(z, t)]$). However, S is not a cut in Δ because the node y has an outgoing edge that is in S and one that is not in S.

However, this can be remedied if all edges carry positive capacity. Then, orthogonality of the current rules out the above possibility.

lemma *cut-RF-separating*:
  fixes Δ (**structure**)
  assumes *sep*: *separating-network* Δ V
  and *sink*: *sink* $\Delta \notin V$
  shows *cut* Δ $(RF^N\ V)$
$\langle proof \rangle$

context
  fixes Δ :: $('v, 'more)$ *network-scheme* **and** Γ (**structure**)
  defines Γ-*def*: $\Gamma \equiv$ *web-of-network* Δ
begin

lemma *separating-network-cut-of-sep*:
  assumes *sep*: *separating* Γ S
  and *source-sink*: *source* $\Delta \neq$ *sink* Δ
  shows *separating-network* Δ (*fst* ‘ $\mathcal{E}$ S)
$\langle proof \rangle$

definition *cut-of-sep* :: $('v \times 'v)$ *set* $\Rightarrow$ $'v$ *set*
where *cut-of-sep* $S = RF^N{}_\Delta$ (*fst* ‘ $\mathcal{E}$ S)

lemma *separating-cut*:
  assumes *sep*: *separating* Γ S
  and *neq*: *source* $\Delta \neq$ *sink* Δ
  and *sink-out*: $\bigwedge x.\ \neg$ *edge* Δ (*sink* Δ) x
  shows *cut* Δ (*cut-of-sep* S)
  $\langle proof \rangle$

context
  fixes f :: $'v$ *edge current* **and** S
  assumes *wf*: *web-flow* Γ f
  and *ortho*: *orthogonal-current* Γ f S
  and *sep*: *separating* Γ S
  and *capacity-pos*: $\bigwedge e.\ e \in \mathbf{E}_\Delta \Longrightarrow$ *capacity* Δ $e > 0$
begin

private lemma *f*: *current* Γ *f* ⟨*proof*⟩

lemma *orthogonal-leave-RF*:
assumes *e*: *edge* Δ *x y*
and *x*: $x \in$ (*cut-of-sep S*)
and *y*: $y \notin$ (*cut-of-sep S*)
shows $(x, y) \in S$
⟨*proof*⟩

lemma *orthogonal-flow-of-current*:
assumes *source-sink*: *source* $\Delta \neq$ *sink* Δ
and *sink-out*: $\bigwedge x.\ \neg$ *edge* Δ (*sink* Δ) *x*
and *no-direct-edge*: $\neg$ *edge* Δ (*source* Δ) (*sink* Δ) — Otherwise, *A* and *B* of the web would not be disjoint.
shows *orthogonal* Δ (*flow-of-current* Δ *f*) (*cut-of-sep S*) (**is** *orthogonal - ?f ?S*)
⟨*proof*⟩

end

end

7.4 Avoiding antiparallel edges and self-loops

context *antiparallel-edges* **begin**

abbreviation *cut′* :: *′a vertex set* ⇒ *′a set* **where** *cut′ S* ≡ *Vertex* −‘ *S*

lemma *cut-cut′*: *cut* Δ'' *S* ⟹ *cut* Δ (*cut′ S*)
⟨*proof*⟩

lemma *IN-Edge*: $\mathbf{IN}_{\Delta''}$ (*Edge x y*) = (*if edge* Δ *x y then* {*Vertex x*} *else* {})
⟨*proof*⟩

lemma *OUT-Edge*: $\mathbf{OUT}_{\Delta''}$ (*Edge x y*) = (*if edge* Δ *x y then* {*Vertex y*} *else* {})
⟨*proof*⟩

interpretation Δ'': *countable-network* Δ'' ⟨*proof*⟩

lemma *d-IN-Edge*:
assumes *f*: *flow* Δ'' *f*
shows *d-IN f* (*Edge x y*) = *f* (*Vertex x*, *Edge x y*)
⟨*proof*⟩

lemma *d-OUT-Edge*:
assumes *f*: *flow* Δ'' *f*
shows *d-OUT f* (*Edge x y*) = *f* (*Edge x y*, *Vertex y*)
⟨*proof*⟩

lemma *orthogonal-cut'*:
assumes *ortho*: *orthogonal* Δ'' *f S*
and *f*: *flow* Δ'' *f*
shows *orthogonal* Δ (*collect f*) (*cut' S*)
⟨*proof*⟩

end

context *countable-network* **begin**

lemma *countable-web-web-of-network*:
assumes *source-in*: $\bigwedge x.\ \neg$ *edge* Δ *x* (*source* Δ)
and *sink-out*: $\bigwedge y.\ \neg$ *edge* Δ (*sink* Δ) *y*
and *undead*: $\bigwedge x\ y.$ *edge* Δ *x y* $\Longrightarrow$ ($\exists z.$ *edge* Δ *y z*) $\vee$ ($\exists z.$ *edge* Δ *z x*)
and *source-sink*: $\neg$ *edge* Δ (*source* Δ) (*sink* Δ)
and *no-loop*: $\bigwedge x.\ \neg$ *edge* Δ *x x*
shows *countable-web* (*web-of-network* Δ) (**is** *countable-web ?*Γ)
⟨*proof*⟩

lemma *max-flow-min-cut'*:
assumes *ex-orthogonal-current*: $\exists f\ S.$ *web-flow* (*web-of-network* Δ) *f* $\wedge$ *separating* (*web-of-network* Δ) *S* $\wedge$ *orthogonal-current* (*web-of-network* Δ) *f S*
and *source-in*: $\bigwedge x.\ \neg$ *edge* Δ *x* (*source* Δ)
and *sink-out*: $\bigwedge y.\ \neg$ *edge* Δ (*sink* Δ) *y*
and *undead*: $\bigwedge x\ y.$ *edge* Δ *x y* $\Longrightarrow$ ($\exists z.$ *edge* Δ *y z*) $\vee$ ($\exists z.$ *edge* Δ *z x*)
and *source-sink*: $\neg$ *edge* Δ (*source* Δ) (*sink* Δ)
and *no-loop*: $\bigwedge x.\ \neg$ *edge* Δ *x x*
and *capacity-pos*: $\bigwedge e.\ e \in \mathbf{E} \Longrightarrow$ *capacity* Δ $e > 0$
shows $\exists f\ S.$ *flow* Δ *f* $\wedge$ *cut* Δ *S* $\wedge$ *orthogonal* Δ *f S*
⟨*proof*⟩

7.5 Eliminating zero edges and incoming edges to *source* and outgoing edges of *sink*

definition Δ''' :: *'v network* **where** Δ''' =
(|*edge* = $\lambda x\ y.$ *edge* Δ *x y* $\wedge$ *capacity* Δ $(x, y) > 0$ $\wedge$ $y \neq$ *source* Δ $\wedge$ $x \neq$ *sink* Δ,
capacity = $\lambda(x, y).$ *if* $x =$ *sink* $\Delta \vee y =$ *source* Δ *then 0 else capacity* Δ (x, y),
source = *source* Δ,
sink = *sink* Δ|)

lemma Δ'''*-sel* [*simp*]:
edge Δ''' *x y* $\longleftrightarrow$ *edge* Δ *x y* $\wedge$ *capacity* Δ $(x, y) > 0$ $\wedge$ $y \neq$ *source* Δ $\wedge$ $x \neq$ *sink* Δ
capacity Δ''' (x, y) = (*if* $x =$ *sink* $\Delta \vee y =$ *source* Δ *then 0 else capacity* Δ (x, y))
source Δ''' = *source* Δ

sink $\Delta''' =$ *sink* Δ
for x y $\langle proof \rangle$

lemma Δ'''*-countable-network*: *countable-network* Δ'''
$\langle proof \rangle$

lemma *flow-*Δ''':
assumes f: *flow* Δ''' f **and** *cut*: *cut* Δ''' S **and** *ortho*: *orthogonal* Δ''' f S
shows *flow* Δ f *cut* Δ S *orthogonal* Δ f S
$\langle proof \rangle$

end

end

8 The max-flow min-cut theorem in bounded networks

8.1 Linkages in unhindered bipartite webs

theory *MFMC-Bounded* **imports**
Matrix-For-Marginals
MFMC-Reduction
begin

context *countable-bipartite-web* **begin**

lemma *countable-A* [*simp*]: *countable* (A Γ)
$\langle proof \rangle$

lemma *unhindered-criterion* [*rule-format*]:
assumes $\neg$ *hindered* Γ
shows $\forall X \subseteq A\ \Gamma.$ *finite* $X \longrightarrow (\sum^+ x{\in}X.\ weight\ \Gamma\ x) \leq (\sum^+ y{\in}\mathbf{E}\ ``\ X.\ weight\ \Gamma\ y)$
$\langle proof \rangle$

end

lemma *nn-integral-count-space-top-approx*:
fixes f :: *nat* => *ennreal* **and** b :: *ennreal*
assumes *nn-integral* (*count-space UNIV*) f = *top*
and b < *top*
obtains n **where** b < *sum* f {..<n}
$\langle proof \rangle$

lemma *One-le-of-nat-ennreal*: (1 :: *ennreal*) $\leq$ *of-nat* $x \longleftrightarrow 1 \leq x$
$\langle proof \rangle$

locale *bounded-countable-bipartite-web* = *countable-bipartite-web* Γ

for Γ :: ($'v$, $'more$) *web-scheme* (**structure**)
+
assumes *bounded-B*: $x \in A\ \Gamma \Longrightarrow (\sum^{+}\ y \in \mathbf{E}\ ``\ \{x\}.\ weight\ \Gamma\ y) < \top$
begin

theorem *unhindered-linkable-bounded*:
assumes $\neg$ *hindered* Γ
shows *linkable* Γ
⟨*proof*⟩

end

8.2 Glueing the reductions together

locale *bounded-countable-web* = *countable-web* Γ
for Γ :: ($'v$, $'more$) *web-scheme* (**structure**)
+
assumes *bounded-out*: $x \in \mathbf{V} - B\ \Gamma \Longrightarrow (\sum^{+}\ y \in \mathbf{E}\ ``\ \{x\}.\ weight\ \Gamma\ y) < \top$
begin

lemma *bounded-countable-bipartite-web-of*: *bounded-countable-bipartite-web* (*bipartite-web-of* Γ)
(**is** *bounded-countable-bipartite-web* *?*Γ)
⟨*proof*⟩

theorem *loose-linkable-bounded*:
assumes *loose* Γ
shows *linkable* Γ
⟨*proof*⟩

lemma *bounded-countable-web-quotient-web*: *bounded-countable-web* (*quotient-web* Γ f) (**is** *bounded-countable-web* *?*Γ)
⟨*proof*⟩

lemma *ex-orthogonal-current*:
$\exists f\ S.$ *web-flow* $\Gamma\ f \wedge$ *separating* $\Gamma\ S \wedge$ *orthogonal-current* $\Gamma\ f\ S$
⟨*proof*⟩

end

locale *bounded-countable-network* = *countable-network* Δ
for Δ :: ($'v$, $'more$) *network-scheme* (**structure**) +
assumes *out*: $[\![\ x \in \mathbf{V};\ x \neq source\ \Delta;\ x \neq sink\ \Delta\ ]\!] \Longrightarrow d\text{-}OUT\ (capacity\ \Delta)\ x < \top$

context *antiparallel-edges* **begin**

lemma Δ''*-bounded-countable-network*: *bounded-countable-network* Δ''
if $\bigwedge x.\ [\![\ x \in \mathbf{V};\ x \neq source\ \Delta;\ x \neq sink\ \Delta\ ]\!] \Longrightarrow d\text{-}OUT\ (capacity\ \Delta)\ x < \top$

⟨*proof*⟩

end

context *bounded-countable-network* **begin**

lemma *bounded-countable-web-web-of-network*:
  assumes *source-in*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ (source\ \Delta)$
  and *sink-out*: $\bigwedge y.\ \neg\ edge\ \Delta\ (sink\ \Delta)\ y$
  and *undead*: $\bigwedge x\ y.\ edge\ \Delta\ x\ y \Longrightarrow (\exists z.\ edge\ \Delta\ y\ z) \vee (\exists z.\ edge\ \Delta\ z\ x)$
  and *source-sink*: $\neg\ edge\ \Delta\ (source\ \Delta)\ (sink\ \Delta)$
  and *no-loop*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ x$
  shows *bounded-countable-web* (*web-of-network* Δ) (**is** *bounded-countable-web ?*Γ)
⟨*proof*⟩

context begin

qualified lemma *max-flow-min-cut′-bounded*:
  assumes *source-in*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ (source\ \Delta)$
  and *sink-out*: $\bigwedge y.\ \neg\ edge\ \Delta\ (sink\ \Delta)\ y$
  and *undead*: $\bigwedge x\ y.\ edge\ \Delta\ x\ y \Longrightarrow (\exists z.\ edge\ \Delta\ y\ z) \vee (\exists z.\ edge\ \Delta\ z\ x)$
  and *source-sink*: $\neg\ edge\ \Delta\ (source\ \Delta)\ (sink\ \Delta)$
  and *no-loop*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ x$
  and *capacity-pos*: $\bigwedge e.\ e \in \mathbf{E} \Longrightarrow capacity\ \Delta\ e > 0$
  shows $\exists f\ S.\ flow\ \Delta\ f \wedge cut\ \Delta\ S \wedge orthogonal\ \Delta\ f\ S$
  ⟨*proof*⟩ **lemma** *max-flow-min-cut′′-bounded*:
  assumes *sink-out*: $\bigwedge y.\ \neg\ edge\ \Delta\ (sink\ \Delta)\ y$
  and *source-in*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ (source\ \Delta)$
  and *no-loop*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ x$
  and *capacity-pos*: $\bigwedge e.\ e \in \mathbf{E} \Longrightarrow capacity\ \Delta\ e > 0$
  shows $\exists f\ S.\ flow\ \Delta\ f \wedge cut\ \Delta\ S \wedge orthogonal\ \Delta\ f\ S$
⟨*proof*⟩ **lemma** *max-flow-min-cut′′′-bounded*:
  assumes *sink-out*: $\bigwedge y.\ \neg\ edge\ \Delta\ (sink\ \Delta)\ y$
  and *source-in*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ (source\ \Delta)$
  and *capacity-pos*: $\bigwedge e.\ e \in \mathbf{E} \Longrightarrow capacity\ \Delta\ e > 0$
  shows $\exists f\ S.\ flow\ \Delta\ f \wedge cut\ \Delta\ S \wedge orthogonal\ \Delta\ f\ S$
⟨*proof*⟩

lemma Δ′′′*-bounded-countable-network*: *bounded-countable-network* Δ′′′
⟨*proof*⟩

theorem *max-flow-min-cut-bounded*:
  $\exists f\ S.\ flow\ \Delta\ f \wedge cut\ \Delta\ S \wedge orthogonal\ \Delta\ f\ S$
⟨*proof*⟩

end

end

end
theory *MFMC-Flow-Attainability* **imports**
MFMC-Network
begin

9 Attainability of flows in networks

9.1 Cleaning up flows

If there is a flow along antiparallel edges, it suffices to consider the difference.

definition $cleanup :: {'a}\ flow \Rightarrow {'a}\ flow$
where $cleanup\ f = (\lambda(a,\ b).\ if\ f\ (a,\ b) > f\ (b,\ a)\ then\ f\ (a,\ b) - f\ (b,\ a)\ else\ 0)$

lemma *cleanup-simps* [*simp*]:
$cleanup\ f\ (a,\ b) = (if\ f\ (a,\ b) > f\ (b,\ a)\ then\ f\ (a,\ b) - f\ (b,\ a)\ else\ 0)$
⟨*proof*⟩

lemma *value-flow-cleanup*:
assumes [*simp*]: $\bigwedge x.\ f\ (x,\ source\ \Delta) = 0$
shows $value\text{-}flow\ \Delta\ (cleanup\ f) = value\text{-}flow\ \Delta\ f$
⟨*proof*⟩

lemma *KIR-cleanup*:
assumes *KIR*: $KIR\ f\ x$
and *finite-IN*: $d\text{-}IN\ f\ x \neq \top$
shows $KIR\ (cleanup\ f)\ x$
⟨*proof*⟩

locale $flow\text{-}attainability = countable\text{-}network\ \Delta$
for $\Delta :: ('v,\ 'more)\ network\text{-}scheme$ (**structure**)
\+
assumes *finite-capacity*: $\bigwedge x.\ x \neq sink\ \Delta \Longrightarrow d\text{-}IN\ (capacity\ \Delta)\ x \neq \top \vee d\text{-}OUT$ $(capacity\ \Delta)\ x \neq \top$
and *no-loop*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ x$
and *source-in*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ (source\ \Delta)$
begin

lemma *source-in-not-cycle*:
assumes $cycle\ \Delta\ p$
shows $(x,\ source\ \Delta) \notin set\ (cycle\text{-}edges\ p)$
⟨*proof*⟩

lemma *source-out-not-cycle*:
$cycle\ \Delta\ p \Longrightarrow (source\ \Delta,\ x) \notin set\ (cycle\text{-}edges\ p)$
⟨*proof*⟩

lemma *flowD-source-IN*:
assumes $flow\ \Delta\ f$

shows *d-IN f* (*source* Δ) = *0*
⟨*proof*⟩

lemma *flowD-finite-IN*:
assumes *f*: *flow* Δ *f* **and** *x*: $x \neq$ *sink* Δ
shows *d-IN f x* ≠ *top*
⟨*proof*⟩

lemma *flowD-finite-OUT*:
assumes *flow* Δ *f x* ≠ *source* Δ *x* ≠ *sink* Δ
shows *d-OUT f x* ≠ ⊤
⟨*proof*⟩

end

locale *flow-network* = *flow-attainability*
+
fixes *g* :: *'v flow*
assumes *g*: *flow* Δ *g*
and *g-finite*: *value-flow* Δ *g* ≠ ⊤
and *nontrivial*: **V** − {*source* Δ, *sink* Δ} ≠ {}
begin

lemma *g-outside*: *e* ∉ **E** ⟹ *g e* = *0*
⟨*proof*⟩

lemma *g-loop* [*simp*]: *g* (*x*, *x*) = *0*
⟨*proof*⟩

lemma *finite-IN-g*: *x* ≠ *sink* Δ ⟹ *d-IN g x* ≠ *top*
⟨*proof*⟩

lemma *finite-OUT-g*:
assumes *x* ≠ *sink* Δ
shows *d-OUT g x* ≠ *top*
⟨*proof*⟩

lemma *g-source-in* [*simp*]: *g* (*x*, *source* Δ) = *0*
⟨*proof*⟩

lemma *finite-g* [*simp*]: *g e* ≠ *top*
⟨*proof*⟩

definition *enum-v* :: *nat* ⇒ *'v*
where *enum-v n* = *from-nat-into* (**V** − {*source* Δ, *sink* Δ}) (*fst* (*prod-decode n*))

lemma *range-enum-v*: *range enum-v* ⊆ **V** − {*source* Δ, *sink* Δ}
⟨*proof*⟩

lemma *enum-v-repeat*:
  assumes x: $x \in \mathbf{V}$ $x \neq \textit{source}\ \Delta$ $x \neq \textit{sink}\ \Delta$
  shows $\exists\, i'{>}i.\ \textit{enum-v}\ i' = x$
$\langle \textit{proof} \rangle$

fun *h-plus* :: $\textit{nat} \Rightarrow {}'v\ \textit{edge} \Rightarrow \textit{ennreal}$
where
  $\textit{h-plus}\ 0\ (x,\ y) = (\textit{if}\ x = \textit{source}\ \Delta\ \textit{then}\ g\ (x,\ y)\ \textit{else}\ 0)$
$|\ \textit{h-plus}\ (\textit{Suc}\ i)\ (x,\ y) =$
  $(\textit{if}\ \textit{enum-v}\ (\textit{Suc}\ i) = x \wedge \textit{d-OUT}\ (\textit{h-plus}\ i)\ x < \textit{d-IN}\ (\textit{h-plus}\ i)\ x\ \textit{then}$
   $\textit{let total} = \textit{d-IN}\ (\textit{h-plus}\ i)\ x - \textit{d-OUT}\ (\textit{h-plus}\ i)\ x;$
       $\textit{share} = g\ (x,\ y) - \textit{h-plus}\ i\ (x,\ y);$
       $\textit{shares} = \textit{d-OUT}\ g\ x - \textit{d-OUT}\ (\textit{h-plus}\ i)\ x$
   $\textit{in h-plus}\ i\ (x,\ y) + \textit{share} * \textit{total}\ /\ \textit{shares}$
   $\textit{else h-plus}\ i\ (x,\ y))$

lemma *h-plus-le-g*: $\textit{h-plus}\ i\ e \leq g\ e$
$\langle \textit{proof} \rangle$

lemma *h-plus-outside*: $e \notin \mathbf{E} \Longrightarrow \textit{h-plus}\ i\ e = 0$
$\langle \textit{proof} \rangle$

lemma *h-plus-not-infty* [*simp*]: $\textit{h-plus}\ i\ e \neq \textit{top}$
$\langle \textit{proof} \rangle$

lemma *h-plus-mono*: $\textit{h-plus}\ i\ e \leq \textit{h-plus}\ (\textit{Suc}\ i)\ e$
$\langle \textit{proof} \rangle$

lemma *h-plus-mono'*: $i \leq j \Longrightarrow \textit{h-plus}\ i\ e \leq \textit{h-plus}\ j\ e$
$\langle \textit{proof} \rangle$

lemma *d-OUT-h-plus-not-infty'*: $x \neq \textit{sink}\ \Delta \Longrightarrow \textit{d-OUT}\ (\textit{h-plus}\ i)\ x \neq \textit{top}$
$\langle \textit{proof} \rangle$

lemma *h-plus-OUT-le-IN*:
  assumes $x \neq \textit{source}\ \Delta$
  shows $\textit{d-OUT}\ (\textit{h-plus}\ i)\ x \leq \textit{d-IN}\ (\textit{h-plus}\ i)\ x$
$\langle \textit{proof} \rangle$

lemma *h-plus-OUT-eq-IN*:
  assumes *enum*: $\textit{enum-v}\ (\textit{Suc}\ i) = x$
  shows $\textit{d-OUT}\ (\textit{h-plus}\ (\textit{Suc}\ i))\ x = \textit{d-IN}\ (\textit{h-plus}\ i)\ x$
$\langle \textit{proof} \rangle$

lemma *h-plus-source-in* [*simp*]: $\textit{h-plus}\ i\ (x,\ \textit{source}\ \Delta) = 0$
$\langle \textit{proof} \rangle$

lemma *h-plus-sum-finite*: $(\sum^+ e.\ h\text{-}plus\ i\ e) \neq top$
⟨*proof*⟩

lemma *d-OUT-h-plus-not-infty* [*simp*]: $d\text{-}OUT\ (h\text{-}plus\ i)\ x \neq top$
⟨*proof*⟩

definition *enum-cycle* :: *nat* ⇒ *'v path*
where *enum-cycle* = *from-nat-into* (*cycles* Δ)

lemma *cycle-enum-cycle* [*simp*]: *cycles* Δ ≠ {} ⟹ *cycle* Δ (*enum-cycle n*)
⟨*proof*⟩

context
  fixes *h'* :: *'v flow*
  assumes *finite-h'*: $h'\ e \neq top$
begin

fun *h-minus-aux* :: *nat* ⇒ *'v edge* ⇒ *ennreal*
where
  h-minus-aux 0 e = *0*
| *h-minus-aux* (*Suc j*) *e* =
  (*if e* ∈ *set* (*cycle-edges* (*enum-cycle j*)) *then*
      h-minus-aux j e + *Min* {*h' e'* − *h-minus-aux j e'*|*e'*. *e'*∈*set* (*cycle-edges*
(*enum-cycle j*))}
   else h-minus-aux j e)

lemma *h-minus-aux-le-h'*: *h-minus-aux j e* ≤ *h' e*
⟨*proof*⟩

lemma *h-minus-aux-finite* [*simp*]: *h-minus-aux j e* ≠ *top*
⟨*proof*⟩

lemma *h-minus-aux-mono*: *h-minus-aux j e* ≤ *h-minus-aux* (*Suc j*) *e*
⟨*proof*⟩

lemma *d-OUT-h-minus-aux*:
  assumes *cycles* Δ ≠ {}
  shows *d-OUT* (*h-minus-aux j*) *x* = *d-IN* (*h-minus-aux j*) *x*
⟨*proof*⟩

lemma *h-minus-aux-source*:
  assumes *cycles* Δ ≠ {}
  shows *h-minus-aux j* (*source* Δ, *y*) = *0*
⟨*proof*⟩

lemma *h-minus-aux-cycle*:
  fixes *j* **defines** *C* ≡ *enum-cycle j*
  assumes *cycles* Δ ≠ {}
  shows ∃ *e*∈*set* (*cycle-edges C*). *h-minus-aux* (*Suc j*) *e* = *h' e*

⟨*proof*⟩

end

fun *h-minus* :: *nat* ⇒ *'v edge* ⇒ *ennreal*
where
  h-minus 0 e = *0*
| *h-minus* (*Suc i*) *e* = *h-minus i e* + (*SUP j*. *h-minus-aux* ($\lambda e'$. *h-plus* (*Suc i*) *e'* − *h-minus i e'*) *j e*)

lemma *h-minus-le-h-plus*: *h-minus i e* ≤ *h-plus i e*
⟨*proof*⟩

lemma *finite-h'*: *h-plus* (*Suc i*) *e* − *h-minus i e* ≠ *top*
  ⟨*proof*⟩

lemma *h-minus-mono*: *h-minus i e* ≤ *h-minus* (*Suc i*) *e*
⟨*proof*⟩

lemma *h-minus-finite* [*simp*]: *h-minus i e* ≠ ⊤
⟨*proof*⟩

lemma *d-OUT-h-minus*:
  assumes *cycles*: *cycles* Δ ≠ {}
  shows *d-OUT* (*h-minus i*) *x* = *d-IN* (*h-minus i*) *x*
⟨*proof*⟩

lemma *h-minus-source*:
  assumes *cycles* Δ ≠ {}
  shows *h-minus n* (*source* Δ, *y*) = *0*
⟨*proof*⟩

lemma *h-minus-source-in* [*simp*]: *h-minus i* (*x*, *source* Δ) = *0*
⟨*proof*⟩

lemma *h-minus-OUT-finite* [*simp*]: *d-OUT* (*h-minus i*) *x* ≠ *top*
⟨*proof*⟩

lemma *h-minus-cycle*:
  assumes *cycle* Δ *C*
  shows ∃ *e*∈*set* (*cycle-edges C*). *h-minus i e* = *h-plus i e*
⟨*proof*⟩

abbreviation *lim-h-plus* :: *'v edge* ⇒ *ennreal*
where *lim-h-plus e* ≡ *SUP n*. *h-plus n e*

abbreviation *lim-h-minus* :: *'v edge* ⇒ *ennreal*
where *lim-h-minus e* ≡ *SUP n*. *h-minus n e*

lemma *lim-h-plus-le-g*: *lim-h-plus e* $\leq$ *g e*
⟨*proof*⟩

lemma *lim-h-plus-finite* [*simp*]: *lim-h-plus e* $\neq$ *top*
⟨*proof*⟩

lemma *lim-h-minus-le-lim-h-plus*: *lim-h-minus e* $\leq$ *lim-h-plus e*
⟨*proof*⟩

lemma *lim-h-minus-finite* [*simp*]: *lim-h-minus e* $\neq$ *top*
⟨*proof*⟩

lemma *lim-h-minus-IN-finite* [*simp*]:
  assumes *x* $\neq$ *sink* Δ
  shows *d-IN lim-h-minus x* $\neq$ *top*
⟨*proof*⟩

lemma *lim-h-plus-OUT-IN*:
  assumes *x* $\neq$ *source* Δ *x* $\neq$ *sink* Δ
  shows *d-OUT lim-h-plus x* = *d-IN lim-h-plus x*
⟨*proof*⟩

lemma *lim-h-minus-OUT-IN*:
  assumes *cycles*: *cycles* $\Delta \neq \{\}$
  shows *d-OUT lim-h-minus x* = *d-IN lim-h-minus x*
⟨*proof*⟩

definition *h* :: *'v edge* $\Rightarrow$ *ennreal*
where *h e* = *lim-h-plus e* − (*if cycles* $\Delta \neq \{\}$ *then lim-h-minus e else 0*)

lemma *h-le-lim-h-plus*: *h e* $\leq$ *lim-h-plus e*
⟨*proof*⟩

lemma *h-le-g*: *h e* $\leq$ *g e*
⟨*proof*⟩

lemma *flow-h*: *flow* Δ *h*
⟨*proof*⟩

lemma *value-h-plus*: *value-flow* Δ (*h-plus i*) = *value-flow* Δ *g* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *value-h*: *value-flow* Δ *h* = *value-flow* Δ *g* (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

definition *h-diff* :: *nat* $\Rightarrow$ *'v edge* $\Rightarrow$ *ennreal*
where *h-diff i e* = *h-plus i e* − (*if cycles* $\Delta \neq \{\}$ *then h-minus i e else 0*)

lemma *d-IN-h-source* [*simp*]: *d-IN* (*h-diff i*) (*source* Δ) = *0*
⟨*proof*⟩

lemma *h-diff-le-h-plus*: *h-diff i e* ≤ *h-plus i e*
⟨*proof*⟩

lemma *h-diff-le-g*: *h-diff i e* ≤ *g e*
⟨*proof*⟩

lemma *h-diff-loop* [*simp*]: *h-diff i* (*x*, *x*) = *0*
⟨*proof*⟩

lemma *supp-h-diff-edges*: *support-flow* (*h-diff i*) ⊆ **E**
⟨*proof*⟩

lemma *h-diff-OUT-le-IN*:
  assumes *x* ≠ *source* Δ
  shows *d-OUT* (*h-diff i*) *x* ≤ *d-IN* (*h-diff i*) *x*
⟨*proof*⟩

lemma *h-diff-cycle*:
  assumes *cycle* Δ *p*
  shows ∃ *e*∈*set* (*cycle-edges p*). *h-diff i e* = *0*
⟨*proof*⟩

lemma *d-IN-h-le-value′*: *d-IN* (*h-diff i*) *x* ≤ *value-flow* Δ (*h-plus i*)
⟨*proof*⟩

lemma *d-IN-h-le-value*: *d-IN h x* ≤ *value-flow* Δ *h* (**is** *?lhs* ≤ *?rhs*)
⟨*proof*⟩

lemma *flow-cleanup*: — Lemma 5.4
  ∃ *h* ≤ *g*. *flow* Δ *h* ∧ *value-flow* Δ *h* = *value-flow* Δ *g* ∧ (∀ *x*. *d-IN h x* ≤ *value-flow* Δ *h*)
⟨*proof*⟩

end

9.2 Residual network

context *countable-network* **begin**

definition *residual-network* :: *′v flow* ⇒ (*′v*, *′more*) *network-scheme*
where *residual-network f* =
  (|*edge* = λ*x y*. *edge* Δ *x y* ∨ *edge* Δ *y x* ∧ *y* ≠ *source* Δ,
   capacity = λ(*x*, *y*). *if edge* Δ *x y then capacity* Δ (*x*, *y*) − *f* (*x*, *y*) *else if y* = *source* Δ *then 0 else f* (*y*, *x*),
   source = *source* Δ, *sink* = *sink* Δ, . . . = *network.more* Δ |)

lemma *residual-network-sel* [*simp*]:
  edge (*residual-network f*) *x y* $\longleftrightarrow$ *edge* Δ *x y* $\vee$ *edge* Δ *y x* $\wedge$ *y* $\neq$ *source* Δ
  capacity (*residual-network f*) (*x*, *y*) = (*if edge* Δ *x y then capacity* Δ (*x*, *y*) − *f* (*x*, *y*) *else if y* = *source* Δ *then 0 else f* (*y*, *x*))
  source (*residual-network f*) = *source* Δ
  sink (*residual-network f*) = *sink* Δ
  network.more (*residual-network f*) = *network.more* Δ
⟨*proof*⟩

lemma E-*residual-network*: $\mathbf{E}_{\textit{residual-network f}} = \mathbf{E} \cup \{(x, y).\ (y, x) \in \mathbf{E} \wedge y \neq$ *source* $\Delta\}$
⟨*proof*⟩

lemma *vertices-residual-network* [*simp*]: *vertex* (*residual-network f*) = *vertex* Δ
⟨*proof*⟩

inductive *wf-residual-network* :: *bool*
where $[\![\ \bigwedge x\ y.\ (x, y) \in \mathbf{E} \Longrightarrow (y, x) \notin \mathbf{E};$ (*source* Δ, *sink* Δ) $\notin \mathbf{E}\ ]\!] \Longrightarrow$ *wf-residual-network*

lemma *wf-residual-networkD*:
  $[\![$ *wf-residual-network*; *edge* Δ *x y* $]\!] \Longrightarrow \neg$ *edge* Δ *y x*
  $[\![$ *wf-residual-network*; $e \in \mathbf{E}$ $]\!] \Longrightarrow$ *prod.swap e* $\notin \mathbf{E}$
  $[\![$ *wf-residual-network*; *edge* Δ (*source* Δ) (*sink* Δ) $]\!] \Longrightarrow$ *False*
⟨*proof*⟩

lemma *residual-countable-network*:
  assumes *wf*: *wf-residual-network*
  and *f*: *flow* Δ *f*
  shows *countable-network* (*residual-network f*) (**is** *countable-network ?*Δ)
⟨*proof*⟩

end

context *antiparallel-edges* **begin**

interpretation Δ'': *countable-network* Δ'' ⟨*proof*⟩

lemma Δ''-*flow-attainability*:
  assumes *flow-attainability-axioms* Δ
  shows *flow-attainability* Δ''
⟨*proof*⟩

lemma Δ''-*wf-residual-network*:
  assumes *no-loop*: $\bigwedge x.\ \neg$ *edge* Δ *x x*
  shows Δ''.*wf-residual-network*
⟨*proof*⟩

end

9.3 The attainability theorem

context *flow-attainability* **begin**

lemma *residual-flow-attainability*:
  assumes *wf*: *wf-residual-network*
  and *f*: *flow* Δ *f*
  shows *flow-attainability* (*residual-network f*) (**is** *flow-attainability ?*Δ)
$\langle$*proof*$\rangle$

end

definition *plus-flow* :: ($'v$, $'more$) *graph-scheme* $\Rightarrow$ $'v$ *flow* $\Rightarrow$ $'v$ *flow* $\Rightarrow$ $'v$ *flow* (**infixr** $\oplus_1$ *65*)
where *plus-flow G f g* = ($\lambda(x, y)$. *if edge G x y then f* $(x, y) + g\ (x, y) - g\ (y, x)$ *else 0*)

lemma *plus-flow-simps* [*simp*]: **fixes** *G* (**structure**) **shows**
  $(f \oplus g)\ (x, y) =$ (*if edge G x y then f* $(x, y) + g\ (x, y) - g\ (y, x)$ *else 0*)
$\langle$*proof*$\rangle$

lemma *plus-flow-outside*: **fixes** *G* (**structure**) **shows** $e \notin \mathbf{E} \Longrightarrow (f \oplus g)\ e = 0$
$\langle$*proof*$\rangle$

lemma
  fixes Δ (**structure**)
  assumes *f-outside*: $\bigwedge e.\ e \notin \mathbf{E} \Longrightarrow f\ e = 0$
  and *g-le-f*: $\bigwedge x\ y.$ *edge* $\Delta\ x\ y \Longrightarrow g\ (y, x) \le f\ (x, y)$
  shows *OUT-plus-flow*: *d-IN* $g\ x \neq$ *top* $\Longrightarrow$ *d-OUT* $(f \oplus g)\ x =$ *d-OUT* $f\ x$ + ($\sum^+ y{\in}UNIV.\ g\ (x, y) *$ *indicator* $\mathbf{E}\ (x, y)$) $-$ ($\sum^+ y.\ g\ (y, x) *$ *indicator* $\mathbf{E}\ (x, y)$)
    (**is** - $\Longrightarrow$ *?OUT* **is** - $\Longrightarrow$ - = - + *?g-out* $-$ *?g-out'*)
  and *IN-plus-flow*: *d-OUT* $g\ x \neq$ *top* $\Longrightarrow$ *d-IN* $(f \oplus g)\ x =$ *d-IN* $f\ x$ + ($\sum^+ y{\in}UNIV.\ g\ (y, x) *$ *indicator* $\mathbf{E}\ (y, x)$) $-$ ($\sum^+ y.\ g\ (x, y) *$ *indicator* $\mathbf{E}\ (y, x)$)
    (**is** - $\Longrightarrow$ *?IN* **is** - $\Longrightarrow$ - = - + *?g-in* $-$ *?g-in'*)
$\langle$*proof*$\rangle$

context *countable-network* **begin**

lemma *d-IN-plus-flow*:
  assumes *wf*: *wf-residual-network*
  and *f*: *flow* Δ *f*
  and *g*: *flow* (*residual-network f*) *g*
  shows *d-IN* $(f \oplus g)\ x \le$ *d-IN* $f\ x$ + *d-IN* $g\ x$
$\langle$*proof*$\rangle$

lemma *scale-flow*:
  assumes *f*: *flow* Δ *f*
  and *c*: $c \le 1$
  shows *flow* Δ ($\lambda e.\ c * f\ e$)

⟨*proof*⟩

lemma *value-scale-flow*:
  value-flow Δ ($\lambda e.\ c * f\ e$) = c * *value-flow* Δ f
⟨*proof*⟩

lemma *value-flow*:
  assumes f: *flow* Δ f
  and *source-out*: $\bigwedge y$. *edge* Δ (*source* Δ) $y \longleftrightarrow y = x$
  shows *value-flow* Δ f = f (*source* Δ, x)
⟨*proof*⟩

end

context *flow-attainability* **begin**

lemma *value-plus-flow*:
  assumes *wf*: *wf-residual-network*
  and f: *flow* Δ f
  and g: *flow* (*residual-network* f) g
  shows *value-flow* Δ ($f \oplus g$) = *value-flow* Δ f + *value-flow* Δ g
⟨*proof*⟩

lemma *flow-residual-add*: — Lemma 5.3
  assumes *wf*: *wf-residual-network*
  and f: *flow* Δ f
  and g: *flow* (*residual-network* f) g
  shows *flow* Δ ($f \oplus g$)
⟨*proof*⟩

definition *minus-flow* :: *'v flow* $\Rightarrow$ *'v flow* $\Rightarrow$ *'v flow* (**infixl** $\ominus$ *65*)
where
  $f \ominus g$ = ($\lambda(x, y)$. *if edge* Δ x y *then* f (x, y) − g (x, y) *else if edge* Δ y x *then* g (y, x) − f (y, x) *else 0*)

lemma *minus-flow-simps* [*simp*]:
  ($f \ominus g$) (x, y) = (*if edge* Δ x y *then* f (x, y) − g (x, y) *else if edge* Δ y x *then* g (y, x) − f (y, x) *else 0*)
⟨*proof*⟩

lemma *minus-flow*:
  assumes *wf*: *wf-residual-network*
  and f: *flow* Δ f
  and g: *flow* Δ g
  and *value-le*: *value-flow* Δ $g \leq$ *value-flow* Δ f
  and *f-finite*: f (*source* Δ, x) $\neq \top$
  and *source-out*: $\bigwedge y$. *edge* Δ (*source* Δ) $y \longleftrightarrow y = x$
  shows *flow* (*residual-network* g) ($f \ominus g$) (**is** *flow ?Δ ?f*)
⟨*proof*⟩

lemma *value-minus-flow*:
  assumes f: *flow* Δ f
  and g: *flow* Δ g
  and *value-le*: *value-flow* Δ $g \leq$ *value-flow* Δ f
  and *source-out*: $\bigwedge y$. *edge* Δ (*source* Δ) $y \longleftrightarrow y = x$
  shows *value-flow* Δ $(f \ominus g) =$ *value-flow* Δ f $-$ *value-flow* Δ g (**is** *?value*)
$\langle proof \rangle$

context
  fixes α
  defines $\alpha \equiv$ (*SUP* $g \in \{g.$ *flow* Δ $g\}$. *value-flow* Δ g)
begin

lemma *flow-by-value*:
  assumes $v < \alpha$
  and *real*[*rule-format*]: $\forall f.\ \alpha = \top \longrightarrow$ *flow* Δ $f \longrightarrow$ *value-flow* Δ $f < \alpha$
  obtains f **where** *flow* Δ f *value-flow* Δ $f = v$
$\langle proof \rangle$

theorem *ex-max-flow'*:
  assumes *wf*: *wf-residual-network*
  assumes *source-out*: $\bigwedge y$. *edge* Δ (*source* Δ) $y \longleftrightarrow y = x$
  and *nontrivial*: $\mathbf{V} - \{$*source* Δ, *sink* $\Delta\} \neq \{\}$
  and *real*: $\alpha =$ *ennreal* α' **and** α'*-nonneg*[*simp*]: $0 \leq \alpha'$
  shows $\exists f.$ *flow* Δ $f \wedge$ *value-flow* Δ $f = \alpha \wedge (\forall x.$ *d-IN* f $x \leq$ *value-flow* Δ $f)$
$\langle proof \rangle$

theorem *ex-max-flow''*: — eliminate assumption of no antiparallel edges using locale *wf-residual-network*
  assumes *source-out*: $\bigwedge y$. *edge* Δ (*source* Δ) $y \longleftrightarrow y = x$
  and *nontrivial*: $\mathbf{E} \neq \{\}$
  and *real*: $\alpha =$ *ennreal* α' **and** *nn*[*simp*]: $0 \leq \alpha'$
  shows $\exists f.$ *flow* Δ $f \wedge$ *value-flow* Δ $f = \alpha \wedge (\forall x.$ *d-IN* f $x \leq$ *value-flow* Δ $f)$
$\langle proof \rangle$

context begin — We eliminate the assumption of only one edge leaving the source by introducing a new source vertex.
private datatype (*plugins del*: *transfer size*) *'v' node* = *SOURCE* | *Inner* (*inner*: *'v'*)

private lemma *not-Inner-conv*: $x \notin$ *range Inner* $\longleftrightarrow x =$ *SOURCE*
$\langle proof \rangle$ **lemma** *inj-on-Inner* [*simp*]: *inj-on Inner A*
$\langle proof \rangle$ **inductive** *edge'* :: *'v node* $\Rightarrow$ *'v node* $\Rightarrow$ *bool*
where
  SOURCE: *edge' SOURCE* (*Inner* (*source* Δ))
| *Inner*: *edge* Δ x $y \Longrightarrow$ *edge'* (*Inner* x) (*Inner* y)

private inductive-simps *edge'-simps* [*simp*]:

$edge'$ $SOURCE$ x
$edge'$ $(Inner\ y)$ x
$edge'$ y $SOURCE$
$edge'$ y $(Inner\ x)$

private fun $capacity'$ :: $'v\ node\ flow$
where
$capacity'$ $(SOURCE, Inner\ x) = (if\ x = source\ \Delta\ then\ \alpha\ else\ 0)$
| $capacity'$ $(Inner\ x, Inner\ y) = capacity\ \Delta\ (x, y)$
| $capacity'$ - $= 0$

private lemma $capacity'$*-source-in* [*simp*]: $capacity'$ $(y, Inner\ (source\ \Delta)) = (if\ y = SOURCE\ then\ \alpha\ else\ 0)$
⟨*proof*⟩ **definition** Δ' :: $'v\ node\ network$
where $\Delta' = (\!|edge = edge', capacity = capacity', source = SOURCE, sink = Inner\ (sink\ \Delta)|\!)$

private lemma Δ'*-sel* [*simp*]:
$edge\ \Delta' = edge'$
$capacity\ \Delta' = capacity'$
$source\ \Delta' = SOURCE$
$sink\ \Delta' = Inner\ (sink\ \Delta)$
⟨*proof*⟩ **lemma** **E**-Δ': $\mathbf{E}_{\Delta'} = \{(SOURCE, Inner\ (source\ \Delta))\} \cup (\lambda(x, y).\ (Inner\ x, Inner\ y))$ ` **E**
⟨*proof*⟩ **lemma** Δ'*-countable-network*:
assumes $\alpha \neq \top$
shows *countable-network* Δ'
⟨*proof*⟩ **lemma** Δ'*-flow-attainability*:
assumes $\alpha \neq \top$
shows *flow-attainability* Δ'
⟨*proof*⟩ **fun** *lift* :: $'v\ flow \Rightarrow 'v\ node\ flow$
where
lift f $(SOURCE, Inner\ y) = (if\ y = source\ \Delta\ then\ value\text{-}flow\ \Delta\ f\ else\ 0)$
| *lift* f $(Inner\ x, Inner\ y) = f\ (x, y)$
| *lift* f - $= 0$

private lemma *d-OUT-lift-Inner* [*simp*]: *d-OUT* (*lift* f) $(Inner\ x)$ = *d-OUT* f x (**is** *?lhs* = *?rhs*)
⟨*proof*⟩ **lemma** *d-OUT-lift-SOURCE* [*simp*]: *d-OUT* (*lift* f) *SOURCE* = *value-flow* Δ f (**is** *?lhs* = *?rhs*)
⟨*proof*⟩ **lemma** *d-IN-lift-Inner* [*simp*]:
assumes $x \neq source\ \Delta$
shows *d-IN* (*lift* f) $(Inner\ x)$ = *d-IN* f x (**is** *?lhs* = *?rhs*)
⟨*proof*⟩ **lemma** *d-IN-lift-source* [*simp*]: *d-IN* (*lift* f) $(Inner\ (source\ \Delta))$ = *value-flow* Δ f + *d-IN* f $(source\ \Delta)$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩ **lemma** *flow-lift* [*simp*]:
assumes *flow* Δ f
shows *flow* Δ' (*lift* f)
⟨*proof*⟩ **abbreviation** (*input*) *unlift* :: $'v\ node\ flow \Rightarrow 'v\ flow$

where *unlift f* $\equiv$ (λ(*x*, *y*). *f* (*Inner x*, *Inner y*))

private lemma *flow-unlift* [*simp*]:
  assumes *f*: *flow* Δ' *f*
  shows *flow* Δ (*unlift f*)
$\langle$*proof*$\rangle$ **lemma** *value-unlift*:
  assumes *f*: *flow* Δ' *f*
  shows *value-flow* Δ (*unlift f*) = *value-flow* Δ' *f*
$\langle$*proof*$\rangle$

theorem *ex-max-flow*:
  $\exists$*f*. *flow* Δ *f* $\wedge$ *value-flow* Δ *f* = α $\wedge$ ($\forall$*x*. *d-IN f x* $\leq$ *value-flow* Δ *f*)
$\langle$*proof*$\rangle$

end

end

end

end

10 The max-flow min-cut theorems in unbounded networks

theory *MFMC-Unbounded* **imports**
  MFMC-Web
  MFMC-Flow-Attainability
  MFMC-Reduction
begin

10.1 More about waves

lemma *SINK-plus-current*: *SINK* (*plus-current f g*) = *SINK f* $\cap$ *SINK g*
$\langle$*proof*$\rangle$

abbreviation *plus-web* :: ('*v*, '*more*) *web-scheme* $\Rightarrow$ '*v current* $\Rightarrow$ '*v current* $\Rightarrow$ '*v current* (- $\frown_1$ - [*66*, *66*] *65*)
where *plus-web* Γ *f g* $\equiv$ *plus-current f* (*g* $\upharpoonright$ Γ / *f*)

lemma *d-OUT-plus-web*:
  fixes Γ (**structure**)
  shows *d-OUT* (*f* $\frown$ *g*) *x* = *d-OUT f x* + *d-OUT* (*g* $\upharpoonright$ Γ / *f*) *x* (**is** *?lhs* = *?rhs*)
$\langle$*proof*$\rangle$

lemma *d-IN-plus-web*:
  fixes Γ (**structure**)
  shows *d-IN* (*f* $\frown$ *g*) *y* = *d-IN f y* + *d-IN* (*g* $\upharpoonright$ Γ / *f*) *y* (**is** *?lhs* = *?rhs*)

⟨*proof*⟩

lemma *plus-web-greater*: $f\ e \le (f \frown_{\Gamma}\ g)\ e$
⟨*proof*⟩

lemma *current-plus-web*:
  fixes Γ (**structure**)
  shows $[\![$ *current* Γ f; *wave* Γ f; *current* Γ g $]\!] \Longrightarrow$ *current* Γ $(f \frown g)$
⟨*proof*⟩

context
  fixes Γ :: ('*v*, '*more*) *web-scheme* (**structure**)
  and $f\ g$ :: '*v current*
  assumes f: *current* Γ f
  and w: *wave* Γ f
  and g: *current* Γ g
begin

context
  fixes x :: '*v*
  assumes x: $x \in \mathcal{E}$ (*TER* f $\cup$ *TER* g)
begin

qualified lemma *RF-f*: $x \notin RF^{\circ}$ (*TER* f)
⟨*proof*⟩ **lemma** *RF-g*: $x \notin RF^{\circ}$ (*TER* g)
⟨*proof*⟩

lemma *TER-plus-web-aux*:
  assumes *SINK*: $x \in$ *SINK* $(g \upharpoonright \Gamma\ /\ f)$ (**is** - $\in$ *SINK ?g*)
  shows $x \in$ *TER* $(f \frown g)$
⟨*proof*⟩ **lemma** *SINK-TER-in''*:
  assumes $\bigwedge x.\ x \notin RF$ (*TER* g) $\Longrightarrow$ *d-OUT* $g\ x = 0$
  shows $x \in$ *SINK* g
⟨*proof*⟩

end

lemma *wave-plus*: *wave* (*quotient-web* Γ f) $(g \upharpoonright \Gamma\ /\ f) \Longrightarrow$ *wave* Γ $(f \frown g)$
⟨*proof*⟩

lemma *TER-plus-web''*:
  assumes $\bigwedge x.\ x \notin RF$ (*TER* g) $\Longrightarrow$ *d-OUT* $g\ x = 0$
  shows $\mathcal{E}$ (*TER* f $\cup$ *TER* g) $\subseteq$ *TER* $(f \frown g)$
⟨*proof*⟩

lemma *TER-plus-web'*: *wave* Γ $g \Longrightarrow \mathcal{E}$ (*TER* f $\cup$ *TER* g) $\subseteq$ *TER* $(f \frown g)$
⟨*proof*⟩

lemma *wave-plus'*: *wave* Γ $g \Longrightarrow$ *wave* Γ $(f \frown g)$

$\langle proof \rangle$

end

lemma *RF-TER-plus-web*:
  fixes Γ (**structure**)
  assumes *f*: *current* Γ *f*
  and *w*: *wave* Γ *f*
  and *g*: *current* Γ *g*
  and *w'*: *wave* Γ *g*
  shows $RF\ (TER\ (f \frown g)) = RF\ (TER\ f \cup TER\ g)$
$\langle proof \rangle$

lemma *RF-TER-Sup*:
  fixes Γ (**structure**)
  assumes *f*: $\bigwedge f.\ f \in Y \Longrightarrow current\ \Gamma\ f$
  and *w*: $\bigwedge f.\ f \in Y \Longrightarrow wave\ \Gamma\ f$
  and *Y*: *Complete-Partial-Order.chain* $(\leq)$ Y $Y \neq \{\}$ *countable* (*support-flow* (*Sup* Y))
  shows $RF\ (TER\ (Sup\ Y)) = RF\ (\bigcup f \in Y.\ TER\ f)$
$\langle proof \rangle$

10.2 Hindered webs with reduced weights

context *countable-bipartite-web* **begin**

context
  fixes $u :: 'v \Rightarrow ennreal$
  and ε
  defines $\varepsilon \equiv (\int^{+} y.\ u\ y\ \partial count\text{-}space\ (B\ \Gamma))$
  assumes *u-outside*: $\bigwedge x.\ x \notin B\ \Gamma \Longrightarrow u\ x = 0$
  and *finite-ε*: $\varepsilon \neq \top$
begin

private lemma *u-A*: $x \in A\ \Gamma \Longrightarrow u\ x = 0$
$\langle proof \rangle$ **lemma** *u-finite*: $u\ y \neq \top$
$\langle proof \rangle$

lemma *hindered-reduce*: — Lemma 6.7
  assumes *u*: $u \leq weight\ \Gamma$
  assumes *hindered-by*: *hindered-by* $(\Gamma(\!|weight := weight\ \Gamma - u|\!))\ \varepsilon$ (**is** *hindered-by* ?Γ -)
  shows *hindered* Γ
$\langle proof \rangle$

end

corollary *hindered-reduce-current*: — Corollary 6.8
  fixes ε g

defines $\varepsilon \equiv \sum^+ x{\in}B\ \Gamma.\ d\text{-}IN\ g\ x - d\text{-}OUT\ g\ x$
assumes *g*: *current* Γ *g*
and *ε-finite*: $\varepsilon \neq \top$
and *hindered*: *hindered-by* $(\Gamma \ominus g)\ \varepsilon$
shows *hindered* Γ
⟨*proof*⟩

end

10.3 Reduced weight in a loose web

definition *reduce-weight* :: ('*v*, '*more*) *web-scheme* ⇒ '*v* ⇒ *real* ⇒ ('*v*, '*more*) *web-scheme*
where *reduce-weight* Γ *x r* = Γ(|*weight* := λ*y*. *weight* Γ *y* − (*if x* = *y then r else 0*)|)

lemma *reduce-weight-sel* [*simp*]:
edge (*reduce-weight* Γ *x r*) = *edge* Γ
A (*reduce-weight* Γ *x r*) = *A* Γ
B (*reduce-weight* Γ *x r*) = *B* Γ
vertex (*reduce-weight* Γ *x r*) = *vertex* Γ
weight (*reduce-weight* Γ *x r*) *y* = (*if x* = *y then weight* Γ *x* − *r else weight* Γ *y*)
web.more (*reduce-weight* Γ *x r*) = *web.more* Γ
⟨*proof*⟩

lemma *essential-reduce-weight* [*simp*]: *essential* (*reduce-weight* Γ *x r*) = *essential* Γ
⟨*proof*⟩

lemma *roofed-reduce-weight* [*simp*]: *roofed-gen* (*reduce-weight* Γ *x r*) = *roofed-gen* Γ
⟨*proof*⟩

context *countable-bipartite-web* **begin**

context begin
private datatype (*plugins del*: *transfer size*) '*a vertex* = *SOURCE* | *SINK* | *Inner* (*inner*: '*a*)

private lemma *notin-range-Inner*: *x* ∉ *range Inner* ⟷ *x* = *SOURCE* ∨ *x* = *SINK*
⟨*proof*⟩ **lemma** *inj-Inner* [*simp*]: ⋀*A*. *inj-on Inner A*
⟨*proof*⟩

lemma *unhinder-bipartite*:
assumes *h*: ⋀*n* :: *nat*. *current* Γ (*h n*)
and *SAT*: ⋀*n*. (*B* Γ ∩ **V**) − {*b*} ⊆ *SAT* Γ (*h n*)
and *b*: *b* ∈ *B* Γ
and *IN*: (*SUP n*. *d-IN* (*h n*) *b*) = *weight* Γ *b*

and *h0-b*: $\bigwedge n.$ *d-IN* $(h\ 0)\ b \leq$ *d-IN* $(h\ n)\ b$
and *b-V*: $b \in \mathbf{V}$
shows $\exists h'.$ *current* $\Gamma\ h' \wedge$ *wave* $\Gamma\ h' \wedge B\ \Gamma \cap \mathbf{V} \subseteq$ *SAT* $\Gamma\ h'$
⟨*proof*⟩

end

lemma *countable-bipartite-web-reduce-weight*:
assumes *weight* $\Gamma\ x \geq w$
shows *countable-bipartite-web* (*reduce-weight* $\Gamma\ x\ w$)
⟨*proof*⟩

lemma *unhinder*: — Lemma 6.9
assumes *loose*: *loose* Γ
and *b*: $b \in B\ \Gamma$
and *wb*: *weight* $\Gamma\ b > 0$
and δ: $\delta > 0$
shows $\exists \varepsilon > 0.\ \varepsilon < \delta \wedge \neg$ *hindered* (*reduce-weight* $\Gamma\ b\ \varepsilon$)
⟨*proof*⟩

end

10.4 Single-vertex saturation in unhindered bipartite webs

The proof of lemma 6.10 in [2] is flawed. The transfinite steps (taking the least upper bound) only preserves unhinderedness, but not looseness. However, the single steps to non-limit ordinals assumes that $\Omega - f_i$ is loose in order to apply Lemma 6.9.

Counterexample: The bipartite web with three nodes a_1, a_2, a_3 in A and two nodes b_1, b_2 in B and edges (a_1, b_1), (a_2, b_1), (a_2, b_2), (a_3, b_2) and weights $a_1 = a_3 = 1$ and $a_2 = 2$ and $b_1 = 3$ and $b_2 = 2$. Then, we can get a sequence of weight reductions on b_2 from 2 to 1.5, 1.25, 1.125, etc. with limit 1. All maximal waves in the restricted webs in the sequence are *zero-current*, so in the limit, we get $k = 0$ and $\varepsilon = 1$ for a_2 and b_2. Now, the restricted web for the two is not loose because it contains the wave which assigns 1 to (a_3, b_2).

We prove a stronger version which only assumes and ensures on unhinderedness.

context *countable-bipartite-web* **begin**

lemma *web-flow-iff*: *web-flow* $\Gamma\ f \longleftrightarrow$ *current* $\Gamma\ f$
⟨*proof*⟩

lemma *countable-bipartite-web-minus-web*:
assumes *f*: *current* $\Gamma\ f$
shows *countable-bipartite-web* $(\Gamma \ominus f)$
⟨*proof*⟩

lemma *current-plus-current-minus*:
assumes f: *current* Γ f
and g: *current* $(\Gamma \ominus f)$ g
shows *current* Γ (*plus-current* f g) (**is** *current* - *?fg*)
⟨*proof*⟩

lemma *wave-plus-current-minus*:
assumes f: *current* Γ f
and w: *wave* Γ f
and g: *current* $(\Gamma \ominus f)$ g
and w': *wave* $(\Gamma \ominus f)$ g
shows *wave* Γ (*plus-current* f g) (**is** *wave* - *?fg*)
⟨*proof*⟩

lemma *minus-plus-current*:
assumes f: *current* Γ f
and g: *current* $(\Gamma \ominus f)$ g
shows $\Gamma \ominus$ *plus-current* f $g = \Gamma \ominus f \ominus g$ (**is** *?lhs* = *?rhs*)
⟨*proof*⟩

lemma *unhindered-minus-web*:
assumes *unhindered*: $\neg$ *hindered* Γ
and f: *current* Γ f
and w: *wave* Γ f
shows $\neg$ *hindered* $(\Gamma \ominus f)$
⟨*proof*⟩

lemma *loose-minus-web*:
assumes *unhindered*: $\neg$ *hindered* Γ
and f: *current* Γ f
and w: *wave* Γ f
and *maximal*: $\bigwedge w.$ ⟦ *current* Γ w; *wave* Γ w; $f \leq w$ ⟧ $\Longrightarrow f = w$
shows *loose* $(\Gamma \ominus f)$ (**is** *loose* *?*Γ)
⟨*proof*⟩

lemma *weight-minus-web*:
assumes f: *current* Γ f
shows *weight* $(\Gamma \ominus f)$ x = (*if* $x \in A$ Γ *then weight* Γ x − *d-OUT* f x *else weight* Γ x − *d-IN* f x)
⟨*proof*⟩

lemma (**in** −) *separating-minus-web* [*simp*]: *separating-gen* $(G \ominus f)$ = *separating-gen* G
⟨*proof*⟩

lemma *current-minus*:
assumes f: *current* Γ f

and *g*: *current* Γ *g*
and *le*: $\bigwedge e.\ g\ e \leq f\ e$
shows *current* $(\Gamma \ominus g)\ (f - g)$
$\langle proof \rangle$

lemma
assumes *w*: *wave* Γ *f*
and *g*: *current* Γ *g*
and *le*: $\bigwedge e.\ g\ e \leq f\ e$
shows *wave-minus*: *wave* $(\Gamma \ominus g)\ (f - g)$
and *TER-minus*: *TER f* $\subseteq$ $TER_{\Gamma \ominus g}\ (f - g)$
$\langle proof \rangle$

lemma (**in** $-$) *essential-minus-web* [*simp*]: *essential* $(\Gamma \ominus f)$ = *essential* Γ
$\langle proof \rangle$

lemma (**in** $-$) *RF-in-essential*: **fixes** *B* **shows** *essential* Γ *B S x* $\Longrightarrow$ $x \in$ *roofed-gen* Γ *B S* $\longleftrightarrow$ $x \in S$
$\langle proof \rangle$

lemma (**in** $-$) *d-OUT-fun-upd*:
assumes $f\ (x, y) \neq \top$ $f\ (x, y) \geq 0$ $k \neq \top$ $k \geq 0$
shows *d-OUT* $(f((x, y) := k))\ x' = ($*if* $x = x'$ *then d-OUT f x* $- f\ (x, y) + k$ *else d-OUT f x'*$)$
(**is** *?lhs = ?rhs*)
$\langle proof \rangle$

lemma *unhindered-saturate1*: — Lemma 6.10
assumes *unhindered*: $\neg$ *hindered* Γ
and *a*: $a \in A\ \Gamma$
shows $\exists f.$ *current* Γ *f* $\wedge$ *d-OUT f a* = *weight* Γ *a* $\wedge$ $\neg$ *hindered* $(\Gamma \ominus f)$
$\langle proof \rangle$

end

10.5 Linkability of unhindered bipartite webs

context *countable-bipartite-web* **begin**

theorem *unhindered-linkable*:
assumes *unhindered*: $\neg$ *hindered* Γ
shows *linkable* Γ
$\langle proof \rangle$

end

context *countable-web* **begin**

theorem *loose-linkable*: — Theorem 6.2
assumes *loose* Γ
shows *linkable* Γ
$\langle proof \rangle$

lemma *ex-orthogonal-current*: — Lemma 4.15
$\exists f\, S.$ *web-flow* $\Gamma\ f \wedge$ *separating* $\Gamma\ S \wedge$ *orthogonal-current* $\Gamma\ f\ S$
$\langle proof \rangle$

end

10.6 Glueing the reductions together

context *countable-network* **begin**

context begin

qualified lemma *max-flow-min-cut'*:
assumes *source-in*: $\bigwedge x.\ \neg$ *edge* $\Delta\ x$ (*source* Δ)
and *sink-out*: $\bigwedge y.\ \neg$ *edge* Δ (*sink* Δ) y
and *undead*: $\bigwedge x\ y.$ *edge* $\Delta\ x\ y \Longrightarrow (\exists z.$ *edge* $\Delta\ y\ z) \vee (\exists z.$ *edge* $\Delta\ z\ x)$
and *source-sink*: $\neg$ *edge* Δ (*source* Δ) (*sink* Δ)
and *no-loop*: $\bigwedge x.\ \neg$ *edge* $\Delta\ x\ x$
and *capacity-pos*: $\bigwedge e.\ e \in \mathbf{E} \Longrightarrow$ *capacity* $\Delta\ e > 0$
shows $\exists f\, S.$ *flow* $\Delta\ f \wedge$ *cut* $\Delta\ S \wedge$ *orthogonal* $\Delta\ f\ S$
$\langle proof \rangle$ **lemma** *max-flow-min-cut''*:
assumes *sink-out*: $\bigwedge y.\ \neg$ *edge* Δ (*sink* Δ) y
and *source-in*: $\bigwedge x.\ \neg$ *edge* $\Delta\ x$ (*source* Δ)
and *no-loop*: $\bigwedge x.\ \neg$ *edge* $\Delta\ x\ x$
and *capacity-pos*: $\bigwedge e.\ e \in \mathbf{E} \Longrightarrow$ *capacity* $\Delta\ e > 0$
shows $\exists f\, S.$ *flow* $\Delta\ f \wedge$ *cut* $\Delta\ S \wedge$ *orthogonal* $\Delta\ f\ S$
$\langle proof \rangle$ **lemma** *max-flow-min-cut'''*:
assumes *sink-out*: $\bigwedge y.\ \neg$ *edge* Δ (*sink* Δ) y
and *source-in*: $\bigwedge x.\ \neg$ *edge* $\Delta\ x$ (*source* Δ)
and *capacity-pos*: $\bigwedge e.\ e \in \mathbf{E} \Longrightarrow$ *capacity* $\Delta\ e > 0$
shows $\exists f\, S.$ *flow* $\Delta\ f \wedge$ *cut* $\Delta\ S \wedge$ *orthogonal* $\Delta\ f\ S$
$\langle proof \rangle$

theorem *max-flow-min-cut*:
$\exists f\, S.$ *flow* $\Delta\ f \wedge$ *cut* $\Delta\ S \wedge$ *orthogonal* $\Delta\ f\ S$
$\langle proof \rangle$

end

end

end

theory *Max-Flow-Min-Cut-Countable* **imports**
  MFMC-Bounded
  MFMC-Unbounded
begin

11 The Max-Flow Min-Cut theorem

theorem *max-flow-min-cut-countable*:
  fixes Δ (**structure**)
  assumes *countable-E* [*simp*]: *countable* **E**
  and *source-neq-sink* [*simp*]: $source\ \Delta \neq sink\ \Delta$
  and *capacity-outside*: $\forall e.\ e \notin \mathbf{E} \longrightarrow capacity\ \Delta\ e = 0$
  and *capacity-finite* [*simp*]: $\forall e.\ capacity\ \Delta\ e \neq \top$
  shows $\exists f\ S.\ flow\ \Delta\ f \wedge cut\ \Delta\ S \wedge orthogonal\ \Delta\ f\ S$
$\langle proof \rangle$

hide-const (**open**) *A B weight*

end

theory *Rel-PMF-Characterisation* **imports**
  Matrix-For-Marginals
begin

12 Characterisation of *rel-pmf*

proposition *rel-pmf-measureI*:
  fixes p :: *'a pmf* **and** q :: *'b pmf*
  assumes *le*: $\bigwedge A.\ measure\ (measure\text{-}pmf\ p)\ A \leq measure\ (measure\text{-}pmf\ q)\ \{y.\ \exists x{\in}A.\ R\ x\ y\}$
  shows *rel-pmf R p q*
$\langle proof \rangle$

12.1 Code generation for *rel-pmf*

proposition *rel-pmf-measureI'*:
  fixes p :: *'a pmf* **and** q :: *'b pmf*
  assumes *le*: $\bigwedge A.\ A \subseteq set\text{-}pmf\ p \Longrightarrow measure\text{-}pmf.prob\ p\ A \leq measure\text{-}pmf.prob\ q\ \{y \in set\text{-}pmf\ q.\ \exists x{\in}A.\ R\ x\ y\}$
  shows *rel-pmf R p q*
$\langle proof \rangle$

lemma *rel-pmf-code* [*code*]:
  $rel\text{-}pmf\ R\ p\ q \longleftrightarrow$
  $(let\ B = set\text{-}pmf\ q\ in$
    $\forall A{\in}Pow\ (set\text{-}pmf\ p).\ measure\text{-}pmf.prob\ p\ A \leq measure\text{-}pmf.prob\ q\ (snd\ `\ Set.filter\ (case\text{-}prod\ R)\ (A \times B)))$

⟨*proof*⟩

end

theory *Rel-PMF-Characterisation-MFMC*
imports
MFMC-Bounded
MFMC-Unbounded
HOL−Library.Simps-Case-Conv
begin

13 Characterisation of *rel-pmf* proved via MFMC

context begin

private datatype ($'a$, $'b$) *vertex* = *Source* | *Sink* | *Left* $'a$ | *Right* $'b$

private lemma *inj-Left* [*simp*]: *inj-on Left X*
⟨*proof*⟩ **lemma** *inj-Right* [*simp*]: *inj-on Right X*
⟨*proof*⟩

context fixes p :: $'a$ *pmf* **and** q :: $'b$ *pmf* **and** R :: $'a \Rightarrow 'b \Rightarrow$ *bool* **begin**

private inductive *edge'* :: ($'a$, $'b$) *vertex* ⇒ ($'a$, $'b$) *vertex* ⇒ *bool* **where**
edge' Source (*Left x*) **if** $x \in$ *set-pmf p*
| *edge'* (*Left x*) (*Right y*) **if** *R x y* $x \in$ *set-pmf p* $y \in$ *set-pmf q*
| *edge'* (*Right y*) *Sink* **if** $y \in$ *set-pmf q*

private inductive-simps *edge'-simps* [*simp*]:
edge' xv (*Left x*)
edge' (*Left x*) (*Right y*)
edge' (*Right y*) *yv*
edge' Source (*Right y*)
edge' Source Sink
edge' xv Source
edge' Sink yv
edge' (*Left x*) *Sink*

private inductive-cases *edge'-SourceE* [*elim*!]: *edge' Source yv*
private inductive-cases *edge'-LeftE* [*elim*!]: *edge'* (*Left x*) *yv*
private inductive-cases *edge'-RightE* [*elim*!]: *edge' xv* (*Right y*)
private inductive-cases *edge'-SinkE* [*elim*!]: *edge' xv Sink*

private function *cap* :: ($'a$, $'b$) *vertex flow* **where**
cap (*xv*, *Left x*) = (*if xv* = *Source then ennreal* (*pmf p x*) *else 0*)
| *cap* (*Left x*, *Right y*) =
(*if R x y* ∧ $x \in$ *set-pmf p* ∧ $y \in$ *set-pmf q*
then pmf q y — Return *pmf q y* so that total weight of x's neighbours is finite,

i.e., the network satisfies *bounded-countable-network*.
  else 0)
| *cap* (*Right y*, *yv*) = (*if yv* = *Sink then ennreal* (*pmf q y*) *else 0*)
| *cap* (*Source*, *Right y*) = *0*
| *cap* (*Source*, *Sink*) = *0*
| *cap* (*xv*, *Source*) = *0*
| *cap* (*Sink*, *yv*) = *0*
| *cap* (*Left x*, *Sink*) = *0*
  ⟨*proof*⟩
termination ⟨*proof*⟩ **definition** Δ :: (*'a*, *'b*) *vertex network*
  where Δ = (|*edge* = *edge'*, *capacity* = *cap*, *source* = *Source*, *sink* = *Sink*|)

private lemma Δ-*sel* [*simp*]:
  edge Δ = *edge'*
  capacity Δ = *cap*
  source Δ = *Source*
  sink Δ = *Sink*
  ⟨*proof*⟩ **lemma** *IN-Left* [*simp*]: $\mathbf{IN}_{\Delta}$ (*Left x*) = (*if x* ∈ *set-pmf p then* {*Source*} *else* {})
  ⟨*proof*⟩ **lemma** *OUT-Right* [*simp*]: $\mathbf{OUT}_{\Delta}$ (*Right y*) = (*if y* ∈ *set-pmf q then* {*Sink*} *else* {})
  ⟨*proof*⟩

interpretation *network*: *countable-network* Δ
⟨*proof*⟩ **lemma** *OUT-cap-Source*: *d-OUT cap Source* = *1*
⟨*proof*⟩ **lemma** *IN-cap-Left*: *d-IN cap* (*Left x*) = *pmf p x*
  ⟨*proof*⟩ **lemma** *OUT-cap-Right*: *d-OUT cap* (*Right y*) = *pmf q y*
  ⟨*proof*⟩ **lemma** *rel-pmf-measureI-aux*:
  assumes *ex-flow*: ∃ *f S*. *flow* Δ *f* ∧ *cut* Δ *S* ∧ *orthogonal* Δ *f S*
    and *le*: ⋀*A*. *measure* (*measure-pmf p*) *A* ≤ *measure* (*measure-pmf q*) {*y*. ∃ *x*∈*A*. *R x y*}
  shows *rel-pmf R p q*
⟨*proof*⟩

proposition *rel-pmf-measureI-unbounded*: — Proof uses the unbounded max-flow min-cut theorem
  assumes *le*: ⋀*A*. *measure* (*measure-pmf p*) *A* ≤ *measure* (*measure-pmf q*) {*y*. ∃ *x*∈*A*. *R x y*}
  shows *rel-pmf R p q*
  ⟨*proof*⟩

interpretation *network*: *bounded-countable-network* Δ
⟨*proof*⟩

proposition *rel-pmf-measureI-bounded*: — Proof uses the bounded max-flow min-cut theorem
  assumes *le*: ⋀*A*. *measure* (*measure-pmf p*) *A* ≤ *measure* (*measure-pmf q*) {*y*. ∃ *x*∈*A*. *R x y*}
  shows *rel-pmf R p q*

⟨*proof*⟩

end

end

interpretation *rel-spmf-characterisation* ⟨*proof*⟩

corollary *rel-pmf-distr-mono*: $\textit{rel-pmf } R\ OO\ \textit{rel-pmf } S \leq \textit{rel-pmf } (R\ OO\ S)$
— This fact has already been proven for the registration of $'a\ pmf$ as a BNF, but this proof is much shorter and more elegant. See [3] for a comparison of formalisations.
⟨*proof*⟩

end

References

[1] R. Aharoni. Menger's theorem for graphs containing no infinite paths. *Europ. J. Combinatorics*, 4:201–204, 1983.

[2] R. Aharoni, E. Berger, A. Georgakopoulos, A. Perlstein, and P. Sprüssel. The max-flow min-cut theorem for countable networks. *J. Combin. Theory Ser. B*, 101:1–17, 2011.

[3] J. Hölzl, A. Lochbihler, and D. Traytel. A formalized hierarchy of probabilistic system types. In C. Urban and X. Zhang, editors, *Interactive Theorem Proving (ITP 2015)*, volume 9236 of *LNCS*, pages 203–220. Springer, 2015.

[4] H. G. Kellerer. Funktionen auf Produkträumen mit vorgegebenen Marginal-Funktionen. *Math. Annalen*, 144:323–344, 1961.