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

Andreas Lochbihler

October 13, 2025

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*
by(*simp add*: *inj-on-def*)

lemma *inj-on-Pair1* [*simp*]: *inj-on* (λx. (x, y)) *A*
by(*simp add*: *inj-on-def*)

lemma *inj-map-prod′*: $[\![$ *inj f*; *inj g* $]\!] \Longrightarrow$ *inj-on* (*map-prod f g*) *X*
by(*rule inj-on-subset*[*OF prod.inj-map subset-UNIV*])

lemma *not-range-Inr*: $x \notin$ *range Inr* $\longleftrightarrow$ $x \in$ *range Inl*
by(*cases x*) *auto*

lemma *not-range-Inl*: $x \notin$ *range Inl* $\longleftrightarrow$ $x \in$ *range Inr*
by(*cases x*) *auto*

lemma *Chains-into-chain*: $M \in$ *Chains* $\{(x, y).\ R\ x\ y\} \Longrightarrow$ *Complete-Partial-Order.chain* *R M*
by(*simp add*: *Chains-def chain-def*)

lemma *chain-dual*: *Complete-Partial-Order.chain* ($\geq$) = *Complete-Partial-Order.chain* ($\leq$)
by(*auto simp add*: *fun-eq-iff chain-def*)

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(*cases* $c > 0$)
case c: *True*
show *?thesis*
proof(*rule metric-CauchyI*)
fix ε :: *real*

assume $0 < \varepsilon$

from *bounded*[*of 0*] x **have** *c-nonneg*: $0 \leq c$ **by** *simp*
from x **have** $0 < ln\ x$ **by** *simp*
from *reals-Archimedean3*[*OF this*] **obtain** $M :: nat$
where $ln\ (c * x) - ln\ (\varepsilon * (x - 1)) < real\ M * ln\ x$ **by** *blast*
hence $exp\ (ln\ (c * x) - ln\ (\varepsilon * (x - 1))) < exp\ (real\ M * ln\ x)$ **by**(*rule exp-less-mono*)
hence M: $c * (1\ /\ x)\ \hat{}\ M\ /\ (1 - 1\ /\ x) < \varepsilon$ **using** ‹$0 < \varepsilon$› x c
by (*simp add*: *exp-diff exp-of-nat-mult field-simps del*: *ln-mult*)

{ fix $n\ m :: nat$
assume $n \geq M$ $m \geq M$
then have $|f\ m - f\ n| \leq c * ((1\ /\ x)\ \hat{}\ M - (1\ /\ x)\ \hat{}\ max\ m\ n)\ /\ (1 - 1\ /\ x)$
proof(*induction n m rule*: *linorder-wlog*)
case *sym* **thus** *?case* **by**(*simp add*: *abs-minus-commute max.commute min.commute*)
next
case (*le m n*)
then show *?case*
proof(*induction* $k \equiv n - m$ *arbitrary*: $n\ m$)
case *0* **thus** *?case* **using** x *c-nonneg* **by**(*simp add*: *field-simps mult-left-mono*)
next
case (*Suc k m n*)
from ‹*Suc k* = -› **obtain** m' **where** m: $m = Suc\ m'$ **by**(*cases m*) *simp-all*
with ‹*Suc k* = -› *Suc.prems* **have** $k = m' - n$ $n \leq m'$ $M \leq n$ $M \leq m'$ **by** *simp-all*
hence $|f\ m' - f\ n| \leq c * ((1\ /\ x)\ \hat{}\ M - (1\ /\ x)\ \hat{}\ (max\ m'\ n))\ /\ (1 - 1\ /\ x)$ **by**(*rule Suc*)
also have $\ldots = c * ((1\ /\ x)\ \hat{}\ M - (1\ /\ x)\ \hat{}\ m')\ /\ (1 - 1\ /\ x)$ **using** ‹$n \leq m'$› **by**(*simp add*: *max-def*)
moreover
from *bounded* **have** $|f\ m - f\ m'| \leq (c\ /\ x\ \hat{}\ m')$ **by**(*simp add*: *m*)
ultimately have $|f\ m' - f\ n| + |f\ m - f\ m'| \leq c * ((1\ /\ x)\ \hat{}\ M - (1\ /\ x)\ \hat{}\ m')\ /\ (1 - 1\ /\ x) + \ldots$ **by** *simp*
also have $\ldots \leq c * ((1\ /\ x)\ \hat{}\ M - (1\ /\ x)\ \hat{}\ m)\ /\ (1 - 1\ /\ x)$ **using** m x **by**(*simp add*: *field-simps*)
also have $|f\ m - f\ n| \leq |f\ m' - f\ n| + |f\ m - f\ m'|$
using *abs-triangle-ineq4*[*of* $f\ m' - f\ n$ $f\ m - f\ m'$] **by** *simp*
ultimately show *?case* **using** ‹$n \leq m$› **by**(*simp add*: *max-def*)
qed
qed
also have $\ldots < c * (1\ /\ x)\ \hat{}\ M\ /\ (1 - 1\ /\ x)$ **using** x c **by**(*auto simp add*: *field-simps*)
also have $\ldots < \varepsilon$ **using** M **.**
finally have $|f\ m - f\ n| < \varepsilon$ **. }**
thus $\exists M.\ \forall m \geq M.\ \forall n \geq M.\ dist\ (f\ m)\ (f\ n) < \varepsilon$ **unfolding** *dist-real-def* **by** *blast*

  qed
next
  case *False*
  with *bounded*[*of 0*] **have** [*simp*]: $c = 0$ **by** *simp*
  have *eq*: $f\ m = f\ 0$ **for** *m*
  proof(*induction m*)
    case (*Suc m*) **from** *bounded*[*of m*] *Suc.IH* **show** *?case* **by** *simp*
  qed *simp*
  show *?thesis* **by**(*rule metric-CauchyI*)(*subst* (*1 2*) *eq*; *simp*)
qed

lemma *complete-lattice-ccpo-dual*:
  class.ccpo Inf ($\geq$) (($>$) :: - :: *complete-lattice* $\Rightarrow$ -)
by(*unfold-locales*)(*auto intro*: *Inf-lower Inf-greatest*)

lemma *card-eq-1-iff*: *card A* = *Suc 0* $\longleftrightarrow$ ($\exists\, x.\ A = \{x\}$)
using *card-eq-SucD* **by** *auto*

lemma *nth-rotate1*: $n <$ *length xs* $\Longrightarrow$ *rotate1 xs* ! n = *xs* ! (*Suc n mod length xs*)
apply(*cases xs*; *clarsimp simp add*: *nth-append not-less*)
apply(*subgoal-tac n* = *length list*; *simp*)
done

lemma *set-zip-rightI*: $[\![$ $x \in$ *set ys*; *length xs* $\geq$ *length ys* $]\!]$ $\Longrightarrow$ $\exists\, z.$ $(z, x) \in$ *set* (*zip xs ys*)
by(*fastforce simp add*: *in-set-zip in-set-conv-nth simp del*: *length-greater-0-conv intro*!: *nth-zip conjI*[*rotated*])

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′*)
by(*auto 4 4 intro*: *exI*[**where** *x*=*take* (*length ys*) *xs*] *exI*[**where** *x*=*drop* (*length ys*) *xs*] *simp add*: *append-eq-conv-conj take-map drop-map dest*: *sym*)

lemma *rotate1-append*:
  rotate1 (*xs* @ *ys*) = (*if xs* = [] *then rotate1 ys else tl xs* @ *ys* @ [*hd xs*])
by(*clarsimp simp add*: *neq-Nil-conv*)

lemma *in-set-tlD*: $x \in$ *set* (*tl xs*) $\Longrightarrow$ $x \in$ *set xs*
by(*cases xs*) *simp-all*

lemma *countable-converseI*:
  assumes *countable A*
  shows *countable* (*converse A*)
proof −
  have *converse A* = *prod.swap* ‘ *A* **by** *auto*
  then show *?thesis* **using** *assms* **by** *simp*
qed

lemma *countable-converse* [*simp*]: *countable* (*converse A*) $\longleftrightarrow$ *countable A*
using *countable-converseI*[*of A*] *countable-converseI*[*of converse A*] **by** *auto*

lemma *nn-integral-count-space-reindex*:
inj-on f A $\Longrightarrow$($\int^+$ *y*. *g y* ∂*count-space* (*f* ' *A*)) = ($\int^+$ *x*. *g* (*f x*) ∂*count-space A*)
by(*simp add*: *embed-measure-count-space'*[*symmetric*] *nn-integral-embed-measure'* *measurable-embed-measure1*)

syntax
-nn-sum :: *pttrn* $\Rightarrow$ *'a set* $\Rightarrow$ *'b* $\Rightarrow$ *'b*::*comm-monoid-add* (‹(2$\sum^+$ -∈-./ -)› [*0*, *51*, *10*] *10*)
-nn-sum-UNIV :: *pttrn* $\Rightarrow$ *'b* $\Rightarrow$ *'b*::*comm-monoid-add* (‹(2$\sum^+$ -./ -)› [*0*, *10*] *10*)
syntax-consts
-nn-sum -nn-sum-UNIV $\rightleftharpoons$ *nn-integral*
translations
$\sum^+$ *i*∈*A*. *b* $\rightleftharpoons$ *CONST nn-integral* (*CONST count-space A*) (λ*i*. *b*)
$\sum^+$ *i*. *b* $\rightleftharpoons$ $\sum^+$ *i*∈*CONST UNIV*. *b*

inductive-simps *rtrancl-path-simps*:
rtrancl-path R x [] *y*
rtrancl-path R x (*a* # *bs*) *y*

definition *restrict-rel* :: *'a set* $\Rightarrow$ (*'a* × *'a*) *set* $\Rightarrow$ (*'a* × *'a*) *set*
where *restrict-rel A R* = {(*x*, *y*)∈*R*. *x* ∈ *A* ∧ *y* ∈ *A*}

lemma *in-restrict-rel-iff*: (*x*, *y*) ∈ *restrict-rel A R* $\longleftrightarrow$ (*x*, *y*) ∈ *R* ∧ *x* ∈ *A* ∧ *y* ∈ *A*
by(*simp add*: *restrict-rel-def*)

lemma *restrict-relE*: ⟦ (*x*, *y*) ∈ *restrict-rel A R*; ⟦ (*x*, *y*) ∈ *R*; *x* ∈ *A*; *y* ∈ *A* ⟧ $\Longrightarrow$ *thesis* ⟧ $\Longrightarrow$ *thesis*
by(*simp add*: *restrict-rel-def*)

lemma *restrict-relI* [*intro*!]: ⟦ (*x*, *y*) ∈ *R*; *x* ∈ *A*; *y* ∈ *A* ⟧ $\Longrightarrow$ (*x*, *y*) ∈ *restrict-rel A R*
by(*simp add*: *restrict-rel-def*)

lemma *Field-restrict-rel-subset*: *Field* (*restrict-rel A R*) ⊆ *A* ∩ *Field R*
by(*auto simp add*: *Field-def in-restrict-rel-iff*)

lemma *Field-restrict-rel* [*simp*]: *Refl R* $\Longrightarrow$ *Field* (*restrict-rel A R*) = *A* ∩ *Field R*
using *Field-restrict-rel-subset*[*of A R*]
by *auto* (*auto simp add*: *Field-def dest*: *refl-onD*)

lemma *Partial-order-restrict-rel*:
assumes *Partial-order R*
shows *Partial-order* (*restrict-rel A R*)

proof −
from *assms* **have** *Refl R* **by**(*simp add*: *order-on-defs*)
from *Field-restrict-rel*[*OF this*, *of A*] *assms* **show** *?thesis*
by(*simp add*: *order-on-defs trans-def antisym-def*)
(*auto intro*: *FieldI1 FieldI2 intro*!: *refl-onI simp add*: *in-restrict-rel-iff elim*: *refl-onD*)
qed

lemma *Chains-restrict-relD*: $M \in$ *Chains* (*restrict-rel A leq*) $\Longrightarrow M \in$ *Chains leq*
by(*auto simp add*: *Chains-def in-restrict-rel-iff*)

lemma *bourbaki-witt-fixpoint-restrict-rel*:
assumes *leq*: *Partial-order leq*
and *chain-Field*: $\bigwedge M.$ ⟦ $M \in$ *Chains* (*restrict-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*
proof
show *Partial-order* (*restrict-rel A leq*) **using** *leq* **by**(*rule Partial-order-restrict-rel*)
next
from *leq* **have** *Refl*: *Refl leq* **by**(*simp add*: *order-on-defs*)

{ **fix** *M z*
assume *M*: $M \in$ *Chains* (*restrict-rel A leq*)
presume *z*: $z \in M$
hence $M \neq \{\}$ **by** *auto*
with *M* **have** *lubA*: *lub* $M \in A$ **by**(*rule chain-Field*)
from *M* **have** *M'*: $M \in$ *Chains leq* **by**(*rule Chains-restrict-relD*)
then have $*$: (z, *lub M*) $\in$ *leq* **using** *z* **by**(*rule lub-upper*)
hence *lub* $M \in$ *Field leq* **by**(*rule FieldI2*)
with *lubA* **show** *lub* $M \in$ *Field* (*restrict-rel A leq*) **by**(*simp add*: *Field-restrict-rel*[*OF Refl*])
from *Chains-FieldD*[*OF M z*] **have** $z \in A$ **by**(*simp add*: *Field-restrict-rel*[*OF Refl*])
with $*$ *lubA* **show** (z, *lub M*) $\in$ *restrict-rel A leq* **by** *auto*

fix *z*
assume *upper*: $\bigwedge x.\ x \in M \Longrightarrow (x, z) \in$ *restrict-rel A leq*
from *upper*[*OF z*] **have** $z \in$ *Field* (*restrict-rel A leq*) **by**(*auto simp add*: *Field-def*)
with *Field-restrict-rel-subset*[*of A leq*] **have** $z \in A$ **by** *blast*
moreover from *lub-least*[*OF M'* ‹$M \neq \{\}$›] *upper* **have** (*lub M*, z) $\in$ *leq*
by(*auto simp add*: *in-restrict-rel-iff*)
ultimately show (*lub M*, z) $\in$ *restrict-rel A leq* **using** *lubA* **by**(*simp add*: *in-restrict-rel-iff*) }
{ **fix** *x*

assume $x \in$ *Field* (*restrict-rel A leq*)
hence $x \in A$ $x \in$ *Field leq* **by**(*simp-all add*: *Field-restrict-rel*[*OF Refl*])
with *increasing*[*OF this*] **show** $(x, f\ x) \in$ *restrict-rel A leq* **by** *auto* **}**
show (*SOME x*. $x \in M$) $\in M$ (*SOME x*. $x \in M$) $\in M$ **if** $M \neq \{\}$ **for** M :: *'a set*
using *that* **by**(*auto intro*: *someI*)
qed

lemma *Field-le* [*simp*]: *Field* $\{(x$ :: - :: *preorder*, $y)$. $x \le y\}$ = *UNIV*
by(*auto intro*: *FieldI1*)

lemma *Field-ge* [*simp*]: *Field* $\{(x$ :: - :: *preorder*, $y)$. $y \le x\}$ = *UNIV*
by(*auto intro*: *FieldI1*)

lemma *refl-le* [*simp*]: *refl* $\{(x$ :: - :: *preorder*, $y)$. $x \le y\}$
by(*auto intro*!: *refl-onI simp add*: *Field-def*)

lemma *refl-ge* [*simp*]: *refl* $\{(x$ :: - :: *preorder*, $y)$. $y \le x\}$
by(*auto intro*!: *refl-onI simp add*: *Field-def*)

lemma *partial-order-le* [*simp*]: *partial-order-on UNIV* $\{(x$ :: - :: *order*, $x')$. $x \le x'\}$
by(*auto simp add*: *order-on-defs trans-def antisym-def*)

lemma *partial-order-ge* [*simp*]: *partial-order-on UNIV* $\{(x$ :: - :: *order*, $x')$. $x' \le x\}$
by(*auto simp add*: *order-on-defs trans-def antisym-def*)

lemma *incseq-chain-range*: *incseq f* $\Longrightarrow$ *Complete-Partial-Order.chain* $(\le)$ (*range f*)
apply(*rule chainI*; *clarsimp*)
using *linear* **by** (*auto dest*: *incseqD*)

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* $x = \{\}$
by(*auto simp add*: *outgoing-def V-def*)

lemma *incoming-outside*: $x \notin V \Longrightarrow$ *incoming* $x = \{\}$
by(*auto simp add*: *incoming-def V-def*)

end

context *NFlow* **begin**

lemma *conservation*: $[\![\ x \neq s;\ x \neq t\ ]\!] \Longrightarrow$ *sum f* (*incoming x*) = *sum f* (*outgoing x*)
by(*cases* $x \in V$)(*auto simp add*: *conservation-const outgoing-outside incoming-outside*)

lemma *augmenting-path-imp-shortest*:
isAugmentingPath p $\Longrightarrow \exists\, p.$ *Graph.isShortestPath cf s p t*
using *Graph.obtain-shortest-path* **unfolding** *isAugmentingPath-def*
by (*fastforce simp*: *Graph.isSimplePath-def Graph.connected-def*)

lemma *shortest-is-augmenting*:
Graph.isShortestPath cf s p t $\Longrightarrow$ *isAugmentingPath p*
unfolding *isAugmentingPath-def* **using** *Graph.shortestPath-is-simple*
by (*fastforce*)

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*
using *assms* **unfolding** *shortest-augmenting-path-def*
by *clarify*(*rule someI-ex*, *rule NFlow.augmenting-path-imp-shortest*)

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

lemma *max-flow*:
NFlow c s t max-flow (**is** *?thesis1*)

¬ (∃ *p*. *NPreflow.isAugmentingPath c s t max-flow p*) (**is** *?thesis2*)
proof −
have *NFlow c s t* (λ-. *0*)
by *unfold-locales*(*simp-all add*: *cap-non-negative*)
moreover have *NFlow c s t* (*NFlow.augment-with-path c f* (*shortest-augmenting-path f*))
if *NFlow c s t f* **and** ∃ *p*. *NPreflow.isAugmentingPath c s t f p* **for** *f*
proof −
interpret *NFlow c s t f* **using** *that*(*1*) **.**
interpret *F*: *Flow c s t NFlow.augment c f* (*NPreflow.augmentingFlow c f* (*shortest-augmenting-path f*))
by(*intro augment-flow-presv augFlow-resFlow shortest-is-augmenting shortest-augmenting-path*[*OF that*])
show *?thesis* **using** *that*
by(*simp add*: *NFlow.augment-with-path-def*)(*unfold-locales*)
qed
ultimately have *NFlow c s t max-flow* ∧ ¬ (∃ *p*. *NPreflow.isAugmentingPath c s t max-flow p*)
unfolding *max-flow-def*
by(*rule while-rule*[**where** *P*=*NFlow c s t* **and** *r*=*measure* (λ*f*. *ek-analysis-defs.ekMeasure* (*residualGraph c f*) *s t*)])
(*auto intro*: *shortest-augmenting-path NFlow.shortest-path-decr-ek-measure simp add*: *NFlow.augment-with-path-def*)
thus *?thesis1 ?thesis2* **by** *simp-all*
qed

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* ⇒ *real*
and *n* :: *nat*
and *R* :: (*nat* × *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* ⇒ *nat* ⇒ *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(*cases* $\exists x{\leq}n.\ f\ x > 0$)
case *False*
hence *f*: $f\ x = 0$ **if** $x \leq n$ **for** *x* **using** *f-nonneg*[*of x*] *that* **by**(*auto simp add*: *not-less*)
hence $sum\ g\ \{..n\} = 0$ **using** *eq-sum* **by** *simp*
hence $g\ y = 0$ **if** $y \leq n$ **for** *y* **using** *g-nonneg that* **by**(*simp add*: *sum-nonneg-eq-0-iff*)
with *f* **show** *thesis* **by**(*auto intro*!: *that*[*of* λ- -. *0*])
next
case *True*
then obtain *x0* **where** *x0*: $x0 \leq n\ f\ x0 > 0$ **by** *blast*

define R' **where** $R' = R \cap \{x.\ f\ x > 0\} \times \{y.\ g\ y > 0\}$

have [*simp*]: *finite* (R ‘‘ A) **for** *A*
proof –
have R ‘‘ $A \subseteq \{..n\}$ **using** *R* **by** *auto*
thus *?thesis* **by**(*rule finite-subset*) *auto*
qed

have R': $R' \subseteq \{..n\} \times \{..n\}$ **using** *R* **by**(*auto simp add*: R'*-def*)
have R'': R' ‘‘ $A \subseteq \{..n\}$ **for** *A* **using** *R* **by**(*auto simp add*: R'*-def*)
have [*simp*]: *finite* (R' ‘‘ A) **for** *A* **using** R''[*of A*]
by(*rule finite-subset*) *auto*

have *hop*: $\exists y0{\leq}n.\ (x0,\ y0) \in R \wedge g\ y0 > 0$ **if** *x0*: $x0 \leq n\ f\ x0 > 0$ **for** *x0*
proof(*rule ccontr*)
assume $\neg$ *?thesis*
then have $g\ y0 = 0$ **if** $(x0,\ y0) \in R\ y0 \leq n$ **for** *y0* **using** *g-nonneg*[*of y0*] *that* **by** *auto*
moreover from *R* **have** R ‘‘ $\{x0\} \subseteq \{..n\}$ **by** *auto*
ultimately have $sum\ g\ (R$ ‘‘ $\{x0\}) = 0$
using *g-nonneg* **by**(*auto intro*!: *sum-nonneg-eq-0-iff*[*THEN iffD2*])
moreover have $\{x0\} \subseteq \{..n\}$ **using** *x0* **by** *auto*
from *le*[*OF this*] *x0* **have** R ‘‘ $\{x0\} \neq \{\}\ sum\ g\ (R$ ‘‘ $\{x0\}) > 0$ **by** *auto*
ultimately show *False* **by** *simp*
qed
then obtain *y0* **where** *y0*: $y0 \leq n\ (x0,\ y0) \in R'\ g\ y0 > 0$ **using** *x0* **by**(*auto simp add*: R'*-def*)

define *LARGE* **where** $LARGE = sum\ f\ \{..n\} + 1$
have $1 \leq LARGE$ **using** *f-nonneg* **by**(*simp add*: *LARGE-def sum-nonneg*)
hence [*simp*]: $LARGE \neq 0\ 0 \neq LARGE\ 0 < LARGE\ 0 \leq LARGE$ **by** *simp-all*

define *s* **where** $s = 2 * n + 2$
define *t* **where** $t = 2 * n + 3$

define c **where** $c = (\lambda(x, y).$
  $\mathit{if}\ x = s \wedge y \leq n\ \mathit{then}\ f\ y$
  $\mathit{else}\ \mathit{if}\ x \leq n \wedge n < y \wedge y \leq 2 * n + 1 \wedge (x, y - n - 1) \in R'\ \mathit{then}\ \mathit{LARGE}$
  $\mathit{else}\ \mathit{if}\ n < x \wedge x \leq 2 * n + 1 \wedge y = t\ \mathit{then}\ g\ (x - n - 1)$
  $\mathit{else}\ 0)$

have *st* [*simp*]: $\neg\ s \leq n\ \neg\ s \leq \mathit{Suc}\ (2 * n)\ s \neq t\ t \neq s\ \neg\ t \leq n\ \neg\ t \leq \mathit{Suc}\ (2 * n)$
  by(*simp-all add*: *s-def t-def*)

have *c-simps*: $c\ (x, y) =$
  $(\mathit{if}\ x = s \wedge y \leq n\ \mathit{then}\ f\ y$
  $\mathit{else}\ \mathit{if}\ x \leq n \wedge n < y \wedge y \leq 2 * n + 1 \wedge (x, y - n - 1) \in R'\ \mathit{then}\ \mathit{LARGE}$
  $\mathit{else}\ \mathit{if}\ n < x \wedge x \leq 2 * n + 1 \wedge y = t\ \mathit{then}\ g\ (x - n - 1)$
  $\mathit{else}\ 0)$
  for $x\ y$ **by**(*simp add*: *c-def*)
have *cs* [*simp*]: $c\ (s, y) = (\mathit{if}\ y \leq n\ \mathit{then}\ f\ y\ \mathit{else}\ 0)$
  and *ct* [*simp*]: $c\ (x, t) = (\mathit{if}\ n < x \wedge x \leq 2 * n + 1\ \mathit{then}\ g\ (x - n - 1)\ \mathit{else}\ 0)$
  for $x\ y$ **by**(*auto simp add*: *c-simps*)

interpret *Graph c* **.**
note [*simp del*] = *zero-cap-simp*

interpret *Network c s t*
proof
  have $(s, x0) \in E$ **using** *x0* **by**(*simp add*: *E-def*)
  thus $s \in V$ **by**(*auto simp add*: *V-def*)

  have $(y0 + n + 1, t) \in E$ **using** *y0* **by**(*simp add*: *E-def*)
  thus $t \in V$ **by**(*auto simp add*: *V-def*)

  show $s \neq t$ **by** *simp*
  show $\forall u\ v.\ 0 \leq c\ (u, v)$ **by**(*simp add*: *c-simps f-nonneg g-nonneg max-def*)
  show $\forall u.\ (u, s) \notin E$ **by**(*simp add*: *E-def c-simps*)
  show $\forall u.\ (t, u) \notin E$ **by**(*simp add*: *E-def c-simps*)
  show $\forall u\ v.\ (u, v) \in E \longrightarrow (v, u) \notin E$ **by**(*simp add*: *E-def c-simps*)

  have *isPath* $s\ [(s, x0), (x0, y0 + n + 1), (y0 + n + 1, t)]\ t$
    using *x0 y0* **by**(*auto simp add*: *E-def c-simps*)
  hence *st*: *connected s t* **by**(*auto simp add*: *connected-def simp del*: *isPath.simps*)

  show $\forall v \in V.$ *connected s v* $\wedge$ *connected v t*
  proof(*intro strip*)
    fix v
    assume v: $v \in V$
    hence $v \leq 2 * n + 3$ **by**(*auto simp add*: *V-def E-def c-simps t-def s-def split*: *if-split-asm*)
    then consider (*left*) $v \leq n$ | (*right*) $n < v\ v \leq 2 * n + 1$ | (s) $v = s$ | (t) $v = t$

by(*fastforce simp add*: *s-def t-def*)
then show *connected s v* $\wedge$ *connected v t*
proof *cases*
case *left*
have *sv*: $(s, v) \in E$ **using** *v left*
by(*fastforce simp add*: *E-def V-def c-simps max-def R′-def split*: *if-split-asm*)
hence *connected s v* **by**(*auto simp add*: *connected-def intro*!: *exI*[**where** $x=[(s, v)]$])
moreover from *sv left f-nonneg*[*of v*] **have** $f\ v > 0$ **by**(*simp add*: *E-def*)
from *hop*[*OF left this*] **obtain** v' **where** $(v, v') \in R$ $v' \leq n$ $g\ v' > 0$ **by** *auto*
hence *isPath v* $[(v, v' + n + 1), (v' + n + 1, t)]$ *t* **using** *left* ‹$f\ v > 0$›
by(*auto simp add*: *E-def c-simps R′-def*)
hence *connected v t* **by**(*auto simp add*: *connected-def simp del*: *isPath.simps*)
ultimately show *?thesis* **..**
next
case *right*
hence *vt*: $(v, t) \in E$ **using** *v* **by**(*auto simp add*: *V-def E-def c-simps max-def R′-def split*: *if-split-asm*)
hence *connected v t* **by**(*auto simp add*: *connected-def intro*!: *exI*[**where** $x=[(v, t)]$])
moreover
have $*$: $g\ (v - n - 1) > 0$ **using** *vt g-nonneg*[*of* $v - n - 1$] *right* **by**(*simp add*: *E-def*)
have $\exists\, v'\leq n.\ (v', v - n - 1) \in R \wedge f\ v' > 0$
proof(*rule ccontr*)
assume $\neg$ *?thesis*
hence *zero*: $[\![\ v' \leq n;\ (v', v - n - 1) \in R\ ]\!] \Longrightarrow f\ v' = 0$ **for** v' **using** *f-nonneg*[*of v′*] **by** *auto*
have *sum f* $\{..n\}$ = *sum f* $\{x.\ x \leq n \wedge x \notin R\hat{}{-1}$ ‘‘ $\{v - n - 1\}\}$
by(*rule sum.mono-neutral-cong-right*)(*auto dest*: *zero*)
also have ... $\leq$ *sum g* (*R* ‘‘ $\{x.\ x \leq n \wedge x \notin R\hat{}{-1}$ ‘‘ $\{v - n - 1\}\}$) **by**(*rule le*) *auto*
also have ... $\leq$ *sum g* $(\{..n\} - \{v - n - 1\})$
by(*rule sum-mono2*)(*use R* **in** ‹*auto simp add*: *g-nonneg*›)
also have ... = *sum g* $\{..n\} - g\ (v - n - 1)$ **using** *right* **by**(*subst sum-diff*) *auto*
also have ... < *sum g* $\{..n\}$ **using** $*$ **by** *simp*
also have ... = *sum f* $\{..n\}$ **by**(*simp add*: *eq-sum*)
finally show *False* **by** *simp*
qed
then obtain v' **where** $v' \leq n$ $(v', v - n - 1) \in R$ $f\ v' > 0$ **by** *auto*
with *right* $*$ **have** *isPath s* $[(s, v'), (v', v)]$ *v* **by**(*auto simp add*: *E-def c-simps R′-def*)
hence *connected s v* **by**(*auto simp add*: *connected-def simp del*: *isPath.simps*)
ultimately show *?thesis* **by** *blast*
qed(*simp-all add*: *st*)
qed

have *reachableNodes s* $\subseteq V$ **using** ‹$s \in V$› **by**(*rule reachable-ss-V*)
also have $V \subseteq \{..2 * n + 3\}$
by(*clarsimp simp add*: *V-def E-def*)(*simp-all add*: *c-simps s-def t-def split*:
if-split-asm)
finally show *finite* (*reachableNodes s*) **by**(*rule finite-subset*) *simp*
qed

interpret *h*: *NFlow c s t max-flow* **by**(*rule max-flow*)
let $?h = \lambda x\ y.\ \mathit{max\text{-}flow}\ (x,\ y + n + 1)$
from *max-flow*(*2*)[*THEN h.fofu-II-III*] **obtain** *C* **where** *C*: *NCut c s t C*
and *eq*: $h.val = NCut.cap\ c\ C$ **by** *blast*
interpret *C*: *NCut c s t C* **using** *C* **.**

have $sum\ c\ (outgoing\ s) = sum\ (\lambda(\text{-},\ x).\ f\ x)\ (Pair\ s\ \text{‘}\ \{..n\})$
by(*rule sum.mono-neutral-cong-left*)(*auto simp add*: *outgoing-def E-def*)
also have $\ldots = sum\ f\ \{..n\}$ **by**(*subst sum.reindex*)(*auto simp add*: *inj-on-def*)
finally have *out*: $sum\ c\ (outgoing\ s) = sum\ f\ \{..n\}$ **.**

have *no-leaving*: $(\lambda y.\ y + n + 1)\ \text{‘}\ (R'\ \text{‘‘}\ (C \cap \{..n\})) \subseteq C$
proof(*rule ccontr*)
assume $\neg\ ?thesis$
then obtain *x y* **where** $*$: $(x,\ y) \in R'\ x \le n\ x \in C\ y + n + 1 \notin C$ **by** *auto*
then have *xy*: $(x,\ y + n + 1) \in E\ y \le n\ c\ (x,\ y + n + 1) = LARGE$
using *R* **by**(*auto simp add*: *E-def c-simps R′-def*)

have $h.val \le sum\ f\ \{..n\}$ **using** *h.val-bounded*(*2*) *out* **by** *simp*
also have $\ldots < sum\ c\ \{(x,\ y + n + 1)\}$ **using** $xy\ *$ **by**(*simp add*: *LARGE-def*)
also have $\ldots \le C.cap$ **using** $*\ xy$ **unfolding** *C.cap-def*
by(*intro sum-mono2*[*OF finite-outgoing′*])(*auto simp add*: *outgoing′-def cap-non-negative*)
also have $\ldots = h.val$ **by**(*simp add*: *eq*)
finally show *False* **by** *simp*
qed

have $C.cap \le sum\ f\ \{..n\}$ **using** *out h.val-bounded*(*2*) *eq* **by** *simp*
then have *cap*: $C.cap = sum\ f\ \{..n\}$
proof(*rule antisym*)
let $?L = \{x.\ x \le n \wedge x \in C \wedge f\ x > 0\}$
let $?R = (\lambda y.\ y + n + 1)\ \text{‘}\ (R'\ \text{‘‘}\ ?L)$
have $sum\ f\ \{..n\} = sum\ f\ ?L + sum\ f\ (\{..n\} - ?L)$ **by**(*subst sum-diff*) *auto*
also have $sum\ f\ ?L \le sum\ g\ (R\ \text{‘‘}\ ?L)$ **by**(*rule le*) *auto*
also have $\ldots = sum\ g\ (R'\ \text{‘‘}\ ?L)$ **using** *g-nonneg*
by(*intro sum.mono-neutral-cong-right*)(*auto 4 3 simp add*: *R′-def Image-iff*
intro: *antisym*)
also have $\ldots = sum\ c\ ((\lambda y.\ (y + n + 1,\ t))\ \text{‘}\ (R'\ \text{‘‘}\ ?L))$ **using** *R*
by(*subst sum.reindex*)(*auto intro*!: *sum.cong simp add*: *inj-on-def R′-def*)
also have $sum\ f\ (\{..n\} - ?L) = sum\ c\ (Pair\ s\ \text{‘}\ (\{..n\} - ?L))$ **by**(*simp add*:
sum.reindex inj-on-def)
also have $sum\ c\ ((\lambda y.\ (y + n + 1,\ t))\ \text{‘}\ (R'\ \text{‘‘}\ ?L)) + \ldots =$
$sum\ c\ (((\lambda y.\ (y + n + 1,\ t))\ \text{‘}\ (R'\ \text{‘‘}\ ?L)) \cup (Pair\ s\ \text{‘}\ (\{..n\} - ?L)))$

by(*subst sum.union-disjoint*) *auto*
also have ... $\leq$ *sum c* $(((\lambda y.\ (y + n + 1,\ t))$ ‘ $(R'$ ‘‘ $?L)) \cup (Pair\ s$ ‘ $(\{..n\} - ?L)) \cup \{(x,\ t) \mid x.\ x \in C \wedge n < x \wedge x \leq 2 * n + 1\})$
by(*rule sum-mono2*)(*auto simp add*: *g-nonneg*)
also have $(((\lambda y.\ (y + n + 1,\ t))$ ‘ $(R'$ ‘‘ $?L)) \cup (Pair\ s$ ‘ $(\{..n\} - ?L)) \cup \{(x,\ t) \mid x.\ x \in C \wedge n < x \wedge x \leq 2 * n + 1\}) = (Pair\ s$ ‘ $(\{..n\} - ?L)) \cup \{(x,\ t) \mid x.\ x \in C \wedge n < x \wedge x \leq 2 * n + 1\}$
using *no-leaving* R' **by**(*fastforce simp add*: *Image-iff intro*: *rev-image-eqI*)
also have *sum c* ... = *sum c* (*outgoing'* C) **using** *C.s-in-cut C.t-ni-cut f-nonneg no-leaving*
apply(*intro sum.mono-neutral-cong-right*)
apply(*auto simp add*: *outgoing'-def E-def intro*: *le-neq-trans*)
apply(*fastforce simp add*: *c-simps Image-iff intro*: *rev-image-eqI split*: *if-split-asm*)+
done
also have ... = *C.cap* **by**(*simp add*: *C.cap-def*)
finally show *sum f* $\{..n\} \leq$ ... **by** *simp*
qed

show *thesis*
proof
show $0 \leq ?h\ x\ y$ **for** $x\ y$ **by**(*rule h.f-non-negative*)
show $(x,\ y) \in R$ **if** $0 < ?h\ x\ y\ x \leq n\ y \leq n$ **for** $x\ y$
using *h.capacity-const*[*rule-format*, *of* $(x,\ y + n + 1)$] *that*
by(*simp add*: *c-simps R'-def split*: *if-split-asm*)

have *sum-h*: *sum* $(?h\ x)\ \{..n\}$ = *max-flow* $(s,\ x)$ **if** $x \leq n$ **for** x
proof −
have *sum* $(?h\ x)\ \{..n\}$ = *sum max-flow* $(Pair\ x$ ‘ $((+)\ (n + 1))$ ‘ $\{..n\})$
by(*simp add*: *sum.reindex add.commute inj-on-def*)
also have ... = *sum max-flow* (*outgoing* x) **using** *that*
apply(*intro sum.mono-neutral-cong-right*)
apply(*auto simp add*: *outgoing-def E-def*)
subgoal for y **by**(*auto 4 3 simp add*: *c-simps max-def split*: *if-split-asm intro*: *rev-image-eqI*[**where** $x=y - n - 1$])
done
also have ... = *sum max-flow* (*incoming* x) **using** *that* **by**(*subst h.conservation*) *auto*
also have ... = *sum max-flow* $\{(s,\ x)\}$ **using** *that*
by(*intro sum.mono-neutral-cong-left*; *auto simp add*: *incoming-def E-def*; *simp add*: *c-simps split*: *if-split-asm*)
finally show *?thesis* **by** *simp*
qed
hence *le*: *sum* $(?h\ x)\ \{..n\} \leq f\ x$ **if** $x \leq n$ **for** x
using *sum-h*[*OF that*] *h.capacity-const*[*rule-format*, *of* $(s,\ x)$] *that* **by** *simp*
moreover have $f\ x \leq$ *sum* $(?h\ x)\ \{..n\}$ **if** $x \leq n$ **for** x
proof(*rule ccontr*)
assume $\neg$ *?thesis*
hence *sum* $(?h\ x)\ \{..n\} < f\ x$ **by** *simp*
hence *sum* $(\lambda x.\ (sum\ (?h\ x)\ \{..n\}))\ \{..n\} <$ *sum f* $\{..n\}$

using *le that* **by**(*intro sum-strict-mono-ex1*) *auto*
also have *sum* ($\lambda x.$ (*sum* (*?h x*) {..*n*})) {..*n*} = *sum max-flow* (*Pair s* ` {..*n*})
using *sum-h* **by**(*simp add*: *sum.reindex inj-on-def*)
also have ... = *sum max-flow* (*outgoing s*)
by(*rule sum.mono-neutral-right*)(*auto simp add*: *outgoing-def E-def*)
also have ... = *sum f* {..*n*} **using** *eq cap* **by**(*simp add*: *h.val-alt*)
finally show *False* **by** *simp*
qed
ultimately show *f x* = *sum* (*?h x*) {..*n*} **if** $x \leq n$ **for** *x* **using** *that* **by**(*auto intro*: *antisym*)

have *sum-h'*: *sum* ($\lambda x.$ *?h x y*) {..*n*} = *max-flow* ($y + n + 1$, *t*) **if** $y \leq n$ **for** *y*
proof −
have *sum* ($\lambda x.$ *?h x y*) {..*n*} = *sum max-flow* (($\lambda x.$ (x, $y + n + 1$)) ` {..*n*})
by(*simp add*: *sum.reindex inj-on-def*)
also have ... = *sum max-flow* (*incoming* ($y + n + 1$)) **using** *that*
apply(*intro sum.mono-neutral-cong-right*)
apply(*auto simp add*: *incoming-def E-def*)
apply(*auto simp add*: *c-simps t-def split*: *if-split-asm*)
done
also have ... = *sum max-flow* (*outgoing* ($y + n + 1$))
using *that* **by**(*subst h.conservation*)(*auto simp add*: *s-def t-def*)
also have ... = *sum max-flow* {($y + n + 1$, *t*)} **using** *that*
by(*intro sum.mono-neutral-cong-left*; *auto simp add*: *outgoing-def E-def*; *simp add*: *s-def c-simps split*: *if-split-asm*)
finally show *?thesis* **by** *simp*
qed
hence *le'*: *sum* ($\lambda x.$ *?h x y*) {..*n*} $\leq$ *g y* **if** $y \leq n$ **for** *y*
using *sum-h'*[*OF that*] *h.capacity-const*[*rule-format*, *of* ($y + n + 1$, *t*)] *that* **by** *simp*
moreover have *g y* $\leq$ *sum* ($\lambda x.$ *?h x y*) {..*n*} **if** $y \leq n$ **for** *y*
proof(*rule ccontr*)
assume ¬ *?thesis*
hence *sum* ($\lambda x.$ *?h x y*) {..*n*} < *g y* **by** *simp*
hence *sum* ($\lambda y.$ (*sum* ($\lambda x.$ *?h x y*) {..*n*})) {..*n*} < *sum g* {..*n*}
using *le' that* **by**(*intro sum-strict-mono-ex1*) *auto*
also have *sum* ($\lambda y.$ (*sum* ($\lambda x.$ *?h x y*) {..*n*})) {..*n*} = *sum max-flow* (($\lambda y.$ ($y + n + 1$, *t*)) ` {..*n*})
using *sum-h'* **by**(*simp add*: *sum.reindex inj-on-def*)
also have ... = *sum max-flow* (*incoming t*)
apply(*rule sum.mono-neutral-right*)
apply *simp*
apply(*auto simp add*: *incoming-def E-def cong*: *rev-conj-cong*)
subgoal for *u* **by**(*auto intro*: *rev-image-eqI*[**where** $x{=}u - n - 1$])
done
also have ... = *sum max-flow* (*outgoing s*) **by**(*rule h.inflow-t-outflow-s*)
also have ... = *sum f* {..*n*} **using** *eq cap* **by**(*simp add*: *h.val-alt*)
finally show *False* **using** *eq-sum* **by** *simp*
qed

ultimately show *g y* = *sum* (λ*x*. *?h x y*) {..*n*} **if** *y* ≤ *n* **for** *y* **using** *that*
by(*auto intro*: *antisym*)
qed
qed

lemma *convergent-bounded-family-nat*:
fixes *f* :: *nat* ⇒ *nat* ⇒ *real*
assumes *bounded*: ⋀*x*. *bounded* (*range* (λ*n*. *f n x*))
obtains *k* **where** *strict-mono k* ⋀*x*. *convergent* (λ*n*. *f* (*k n*) *x*)
proof −
interpret *subseqs* λ*x k*. *convergent* (λ*n*. *f* (*k n*) *x*)
proof(*unfold-locales*)
fix *x* **and** *s* :: *nat* ⇒ *nat*
have *bounded* (*range* (λ*n*. *f* (*s n*) *x*)) **using** *bounded* **by**(*rule bounded-subset*)
auto
from *bounded-imp-convergent-subsequence*[*OF this*]
show ∃ *r*. *strict-mono r* ∧ *convergent* (λ*n*. *f* ((*s* ∘ *r*) *n*) *x*)
by(*auto simp add*: *o-def convergent-def*)
qed
{ fix *k*
have *convergent* (λ*n*. *f* ((*diagseq* ∘ (+) (*Suc k*)) *n*) *k*)
by(*rule diagseq-holds*)(*auto dest*: *convergent-subseq-convergent simp add*: *o-def*)
hence *convergent* (λ*n*. *f* (*diagseq n*) *k*) **unfolding** *o-def*
by(*subst* (*asm*) *add.commute*)(*simp only*: *convergent-ignore-initial-segment*[**where**
f=λ*x*. *f* (*diagseq x*) *k*])
} **with** *subseq-diagseq* **show** *?thesis* **..**
qed

lemma *convergent-bounded-family*:
fixes *f* :: *nat* ⇒ ′*a* ⇒ *real*
assumes *bounded*: ⋀*x*. *x* ∈ *A* ⟹ *bounded* (*range* (λ*n*. *f n x*))
and *A*: *countable A*
obtains *k* **where** *strict-mono k* ⋀*x*. *x* ∈ *A* ⟹ *convergent* (λ*n*. *f* (*k n*) *x*)
proof(*cases A* = {})
case *False*
define *f*′ **where** *f*′ *n x* = *f n* (*from-nat-into A x*) **for** *n x*
have *bounded* (*range* (λ*n*. *f*′ *n x*)) **for** *x*
unfolding *f*′*-def* **using** *from-nat-into*[*OF False*] **by**(*rule bounded*)
then obtain *k* **where** *k*: *strict-mono k*
and *conv*: *convergent* (λ*n*. *f*′ (*k n*) *x*) **for** *x*
by(*rule convergent-bounded-family-nat*) *iprover*
have *convergent* (λ*n*. *f* (*k n*) *x*) **if** *x* ∈ *A* **for** *x*
using *conv*[*of to-nat-on A x*] *A that* **by**(*simp add*: *f*′*-def*)
with *k* **show** *thesis* **..**
next
case *True*
with *strict-mono-id* **show** *thesis* **by**(*blast intro*!: *that*)
qed

```
abbreviation zero-on :: ('a ⇒ 'b :: zero) ⇒ 'a set ⇒ 'a ⇒ 'b
where zero-on f ≡ override-on f (λ-. 0)

lemma zero-on-le [simp]: fixes f :: 'a ⇒ 'b :: {preorder, zero} shows
  zero-on f X x ≤ f x ⟷ (x ∈ X ⟶ 0 ≤ f x)
by(auto simp add: override-on-def)

lemma zero-on-nonneg: fixes f :: 'a ⇒ 'b :: {preorder, zero} shows
  0 ≤ zero-on f X x ⟷ (x ∉ X ⟶ 0 ≤ f x)
by(auto simp add: override-on-def)

lemma sums-zero-on:
  fixes f :: nat ⇒ 'a::real-normed-vector
  assumes f: f sums s
    and X: finite X
  shows zero-on f X sums (s − sum f X)
proof −
  have (λx. if x ∈ X then f x else 0) sums sum (λx. f x) X using X by(rule
sums-If-finite-set)
  from sums-diff[OF f this] show ?thesis
    by(simp add: sum-negf override-on-def if-distrib cong del: if-weak-cong)
qed

lemma
  fixes f :: nat ⇒ 'a::real-normed-vector
  assumes f: summable f
  and X: finite X
  shows summable-zero-on [simp]: summable (zero-on f X) (is ?thesis1)
  and suminf-zero-on: suminf (zero-on f X) = suminf f − sum f X (is ?thesis2)
proof −
  from f obtain s where f sums s unfolding summable-def ..
  with sums-zero-on[OF this X] show ?thesis1 ?thesis2
    by(auto simp add: summable-def sums-unique[symmetric])
qed

lemma summable-zero-on-nonneg:
 fixes f :: nat ⇒ 'a :: {ordered-comm-monoid-add,linorder-topology,conditionally-complete-linorder}
  assumes f: summable f
  and nonneg: ⋀x. 0 ≤ f x
  shows summable (zero-on f X)
proof(rule summableI-nonneg-bounded)
  fix n
  have sum (zero-on f X) {..<n} ≤ sum f {..<n} by(rule sum-mono)(simp add:
nonneg)
  also have ... ≤ suminf f using f by(rule sum-le-suminf)(auto simp add: non-
neg)
  finally show sum (zero-on f X) {..<n} ≤ suminf f .
qed(simp add: zero-on-nonneg nonneg)
```

lemma *zero-on-ennreal* [*simp*]: *zero-on* (λx. *ennreal* (*f x*)) *A* = (λx. *ennreal* (*zero-on f A x*))
by(*simp add*: *override-on-def fun-eq-iff*)

lemma *sum-lessThan-conv-atMost-nat*:
  fixes *f* :: *nat* ⇒ *'b* :: *ab-group-add*
  shows *sum f* {..<*n*} = *sum f* {..*n*} − *f n*
by (*metis Groups.add-ac*(*2*) *add-diff-cancel-left′ lessThan-Suc-atMost sum.lessThan-Suc*)

lemma *Collect-disjoint-atLeast*:
  Collect P ∩ {*x*..} = {} ⟷ (∀ *y*≥*x*. ¬ *P y*)
by(*auto simp add*: *atLeast-def*)

lemma *bounded-matrix-for-marginals-nat*:
  fixes *f g* :: *nat* ⇒ *real*
    and *R* :: (*nat* × *nat*) *set*
    and *s* :: *real*
  assumes *sum-f*: *f sums s* **and** *sum-g*: *g sums s*
    and *f-nonneg*: ⋀*x*. *0* ≤ *f x* **and** *g-nonneg*: ⋀*y*. *0* ≤ *g y*
    and *f-le-g*: ⋀*X*. *suminf* (*zero-on f* (− *X*)) ≤ *suminf* (*zero-on g* (− *R* ‘‘ *X*))
  obtains *h* :: *nat* ⇒ *nat* ⇒ *real*
    where ⋀*x y*. *0* ≤ *h x y*
    and ⋀*x y*. *0* < *h x y* ⟹ (*x*, *y*) ∈ *R*
    and ⋀*x*. *h x sums f x*
    and ⋀*y*. (λx. *h x y*) *sums g y*
proof −
  have *summ-f*: *summable f* **and** *summ-g*: *summable g* **and** *sum-fg*: *suminf f* = *suminf g*
    using *sum-f sum-g* **by**(*auto simp add*: *summable-def sums-unique*[*symmetric*])

  have *summ-zf*: *summable* (*zero-on f A*) **for** *A*
    using *summ-f f-nonneg* **by**(*rule summable-zero-on-nonneg*)
  have *summ-zg*: *summable* (*zero-on g A*) **for** *A*
    using *summ-g g-nonneg* **by**(*rule summable-zero-on-nonneg*)

  define *f′* :: *nat* ⇒ *nat* ⇒ *real*
    where *f′ n x* = (*if x* ≤ *n then f x else if x* = *Suc n then* ∑ *k*. *f* (*k* + (*n* + *1*)) *else 0*) **for** *n x*
  define *g′* :: *nat* ⇒ *nat* ⇒ *real*
    where *g′ n y* = (*if y* ≤ *n then g y else if y* = *Suc n then* ∑ *k*. *g* (*k* + (*n* + *1*)) *else 0*) **for** *n y*
  define *R′* :: *nat* ⇒ (*nat* × *nat*) *set*
    where *R′ n* =
      R ∩ {..*n*} × {..*n*} ∪
      {(*n* + *1*, *y*) | *y*. ∃ *x′*>*n*. (*x′*, *y*) ∈ *R* ∧ *y* ≤ *n*} ∪
      {(*x*, *n* + *1*) | *x*. ∃ *y′*>*n*. (*x*, *y′*) ∈ *R* ∧ *x* ≤ *n*} ∪
      (*if* ∃ *x*>*n*. ∃ *y*>*n*. (*x*, *y*) ∈ *R then* {(*n* + *1*, *n* + *1*)} *else* {})
    for *n*
  have *R′-simps* [*simplified*, *simp*]:

$[\![x \leq n; y \leq n]\!] \Longrightarrow (x, y) \in R' \ n \longleftrightarrow (x, y) \in R$
$y \leq n \Longrightarrow (n + 1, y) \in R' \ n \longleftrightarrow (\exists x'>n. (x', y) \in R)$
$x \leq n \Longrightarrow (x, n + 1) \in R' \ n \longleftrightarrow (\exists y'>n. (x, y') \in R)$
$(n + 1, n + 1) \in R' \ n \longleftrightarrow (\exists x'>n. \exists y'>n. (x', y') \in R)$
$x > n + 1 \vee y > n + 1 \Longrightarrow (x, y) \notin R' \ n$
for *x y n* **by**(*auto simp add*: *R'-def*)

have *R'-cases*: *thesis* **if** $(x, y) \in R' \ n$
and $[\![x \leq n; y \leq n; (x, y) \in R]\!] \Longrightarrow thesis$
and $\bigwedge x'. [\![x = n + 1; y \leq n; n < x'; (x', y) \in R]\!] \Longrightarrow thesis$
and $\bigwedge y'. [\![x \leq n; y = n + 1; n < y'; (x, y') \in R]\!] \Longrightarrow thesis$
and $\bigwedge x' \ y'. [\![x = n + 1; y = n + 1; n < x'; n < y'; (x', y') \in R]\!] \Longrightarrow thesis$
for *thesis x y n* **using** *that* **by**(*auto simp add*: *R'-def split*: *if-split-asm*)

have *R'-intros*:
$[\![(x, y) \in R; x \leq n; y \leq n]\!] \Longrightarrow (x, y) \in R' \ n$
$[\![(x', y) \in R; n < x'; y \leq n]\!] \Longrightarrow (n + 1, y) \in R' \ n$
$[\![(x, y') \in R; x \leq n; n < y']\!] \Longrightarrow (x, n + 1) \in R' \ n$
$[\![(x', y') \in R; n < x'; n < y']\!] \Longrightarrow (n + 1, n + 1) \in R' \ n$
for *n x y x' y'* **by**(*auto*)

have *Image-R'*:
$R' \ n \ `` \ X = (R \ `` \ (X \cap \{..n\})) \cap \{..n\} \cup$
$(if \ n + 1 \in X \ then \ (R \ `` \ \{n+1..\}) \cap \{..n\} \ else \ \{\}) \cup$
$(if \ (R \ `` \ (X \cap \{..n\})) \cap \{n+1..\} = \{\} \ then \ \{\} \ else \ \{n + 1\}) \cup$
$(if \ n + 1 \in X \wedge (R \ `` \ \{n+1..\}) \cap \{n+1..\} \neq \{\} \ then \ \{n + 1\} \ else \ \{\})$ **for** *n*
X
apply(*simp add*: *Image-def*)
apply(*safe elim*!: *R'-cases*; *auto simp add*: *Collect-disjoint-atLeast intro*: *R'-intros*
simp add: *Suc-le-eq dest*: *Suc-le-lessD*)
apply(*metis R'-simps*(*4*) *R'-intros*(*3*) *Suc-eq-plus1*)+
done

{ fix *n*
have $sum \ (f' \ n) \ \{..n + 1\} = sum \ (g' \ n) \ \{..n + 1\}$ **using** *sum-fg*
unfolding *f'-def g'-def suminf-minus-initial-segment*[*OF summ-f*] *suminf-minus-initial-segment*[*OF*
summ-g]
by(*simp*)(*metis* (*no-types*, *opaque-lifting*) *add.commute add.left-inverse atLeast0At-*
Most atLeast0LessThan atLeastLessThanSuc-atLeastAtMost minus-add-distrib sum.lessThan-Suc
uminus-add-conv-diff)
moreover have $sum \ (f' \ n) \ X \leq sum \ (g' \ n) \ (R' \ n \ `` \ X)$ **if** $X \subseteq \{..n + 1\}$ **for**
X
proof −
from *that* **have** [*simp*]: *finite X* **by**(*rule finite-subset*) *simp*
define *X'* **where** $X' \equiv if \ n + 1 \in X \ then \ X \cup \{n+1..\} \ else \ X$

define *Y'* **where** $Y' \equiv if \ R \ `` \ X' \cap \{n+1..\} = \{\} \ then \ R \ `` \ X' \ else \ R \ `` \ X'$
$\cup \{n+1..\}$

have *sum* $(f'\ n)\ X = sum\ (f'\ n)\ (X - \{n + 1\}) + (if\ n + 1 \in X\ then\ f'\ n\ (n + 1)\ else\ 0)$
by(*simp add*: *sum.remove*)
also have *sum* $(f'\ n)\ (X - \{n + 1\}) = sum\ f\ (X - \{n + 1\})$ **using** *that*
by(*intro sum.cong*)(*auto simp add*: f'*-def*)
also have $\ldots + (if\ n + 1 \in X\ then\ f'\ n\ (n + 1)\ else\ 0) = suminf\ (zero\text{-}on\ f\ (- X'))$
proof(*cases* $n + 1 \in X$)
case *True*
with *sum-f that* **show** *?thesis*
apply(*simp add*: *summable-def* X'*-def* f'*-def suminf-zero-on*[*OF sums-summable*] *del*: *One-nat-def*)
apply(*subst suminf-minus-initial-segment*[*OF summ-f*])
apply(*simp add*: *algebra-simps*)
apply(*subst sum.union-disjoint*[*symmetric*])
apply(*auto simp add*: *sum-lessThan-conv-atMost-nat intro*!: *sum.cong*)
done
next
case *False*
with *sum-f* **show** *?thesis*
by(*simp add*: X'*-def suminf-finite*[**where** $N{=}X$])
qed
also have $\ldots \le suminf\ (zero\text{-}on\ g\ (- R\ `` X'))$ **by**(*rule f-le-g*)
also have $\ldots \le suminf\ (zero\text{-}on\ g\ (- Y'))$
by(*rule suminf-le*[*OF - summ-zg summ-zg*])(*clarsimp simp add*: *override-on-def g-nonneg* Y'*-def summ-zg*)
also have $\ldots = suminf\ (\lambda k.\ zero\text{-}on\ g\ (- Y')\ (k + (n + 1))) + sum\ (zero\text{-}on\ g\ (- Y'))\ \{..n\}$
by(*subst suminf-split-initial-segment*[*OF summ-zg, of - n + 1*])(*simp add*: *sum-lessThan-conv-atMost-nat*)
also have *sum* $(zero\text{-}on\ g\ (- Y'))\ \{..n\} = sum\ g\ (Y' \cap \{..n\})$
by(*rule sum.mono-neutral-cong-right*)(*auto simp add*: *override-on-def*)
also have $\ldots = sum\ (g'\ n)\ (Y' \cap \{..n\})$
by(*rule sum.cong*)(*auto simp add*: g'*-def*)
also have *suminf* $(\lambda k.\ zero\text{-}on\ g\ (- Y')\ (k + (n + 1))) \le (if\ R\ `` X' \cap \{n{+}1..\} = \{\}\ then\ 0\ else\ g'\ n\ (n + 1))$
apply(*clarsimp simp add*: Y'*-def* g'*-def simp del*: *One-nat-def*)
apply(*subst suminf-eq-zero-iff*[*THEN iffD2*])
apply(*auto simp del*: *One-nat-def simp add*: *summable-iff-shift summ-zg zero-on-nonneg g-nonneg*)
apply(*auto simp add*: *override-on-def*)
done
also have $\ldots + sum\ (g'\ n)\ (Y' \cap \{..n\}) = sum\ (g'\ n)\ (R'\ n\ `` X)$
using *that* **by**(*fastforce simp add*: *Image-R'* Y'*-def* X'*-def atMost-Suc intro*: *sum.cong*[*OF - refl*])
finally show *?thesis* **by** *simp*
qed
moreover have $\bigwedge x.\ 0 \le f'\ n\ x\ \bigwedge y.\ 0 \le g'\ n\ y$
by(*auto simp add*: f'*-def* g'*-def f-nonneg g-nonneg summable-iff-shift summ-f*

summ-g intro!: *suminf-nonneg simp del*: *One-nat-def*)
  moreover have R' $n \subseteq \{..n+1\} \times \{..n+1\}$
   by(*auto elim*!: R'*-cases*)
  ultimately obtain h
   where $\bigwedge x\ y.\ [\![\ x \le n + 1;\ y \le n + 1]\!] \Longrightarrow 0 \le h\ x\ y$
   and $\bigwedge x\ y.\ [\![\ 0 < h\ x\ y;\ x \le n + 1;\ y \le n + 1]\!] \Longrightarrow (x,\ y) \in R'\ n$
   and $\bigwedge x.\ x \le n + 1 \Longrightarrow f'\ n\ x = sum\ (h\ x)\ \{..n + 1\}$
   and $\bigwedge y.\ y \le n + 1 \Longrightarrow g'\ n\ y = sum\ (\lambda x.\ h\ x\ y)\ \{..n + 1\}$
   by(*rule bounded-matrix-for-marginals-finite*) *blast*+
  hence $\exists h.\ (\forall x\ y.\ x \le n + 1 \longrightarrow y \le n + 1 \longrightarrow 0 \le h\ x\ y) \wedge$
   $(\forall x\ y.\ 0 < h\ x\ y \longrightarrow x \le n + 1 \longrightarrow y \le n + 1 \longrightarrow (x,\ y) \in R'\ n) \wedge$
   $(\forall x.\ x \le n + 1 \longrightarrow f'\ n\ x = sum\ (h\ x)\ \{..n + 1\}) \wedge$
   $(\forall y.\ y \le n + 1 \longrightarrow g'\ n\ y = sum\ (\lambda x.\ h\ x\ y)\ \{..n + 1\})$ **by** *blast* **}**
 hence $\exists h.\ \forall n.\ (\forall x\ y.\ x \le n + 1 \longrightarrow y \le n + 1 \longrightarrow 0 \le h\ n\ x\ y) \wedge$
   $(\forall x\ y.\ 0 < h\ n\ x\ y \longrightarrow x \le n + 1 \longrightarrow y \le n + 1 \longrightarrow (x,\ y) \in R'\ n) \wedge$
   $(\forall x.\ x \le n + 1 \longrightarrow f'\ n\ x = sum\ (h\ n\ x)\ \{..n + 1\}) \wedge$
   $(\forall y.\ y \le n + 1 \longrightarrow g'\ n\ y = sum\ (\lambda x.\ h\ n\ x\ y)\ \{..n + 1\})$
  by(*subst choice-iff*[*symmetric*]) *blast*
 then obtain h **where** *h-nonneg*: $\bigwedge n\ x\ y.\ [\![\ x \le n + 1;\ y \le n + 1]\!] \Longrightarrow 0 \le h$
$n\ x\ y$
  and *h-R*: $\bigwedge n\ x\ y.\ [\![\ 0 < h\ n\ x\ y;\ x \le n + 1;\ y \le n + 1]\!] \Longrightarrow (x,\ y) \in R'\ n$
  and *h-f*: $\bigwedge n\ x.\ x \le n + 1 \Longrightarrow f'\ n\ x = sum\ (h\ n\ x)\ \{..n + 1\}$
  and *h-g*: $\bigwedge n\ y.\ y \le n + 1 \Longrightarrow g'\ n\ y = sum\ (\lambda x.\ h\ n\ x\ y)\ \{..n + 1\}$
  apply(*rule exE*)
  subgoal for h **by**(*erule meta-allE*[*of - h*]) *blast*
  done

 define h' :: $nat \Rightarrow nat \times nat \Rightarrow real$
  where $h'\ n = (\lambda(x,\ y).\ if\ x \le n \wedge y \le n\ then\ h\ n\ x\ y\ else\ 0)$ **for** n
 have h'*-nonneg*: $h'\ n\ xy \ge 0$ **for** $n\ xy$ **by**(*simp add*: h'*-def h-nonneg split*:
prod.split)

 have $h'\ n\ xy \le s$ **for** $n\ xy$
 proof(*cases xy*)
  case [*simp*]: (*Pair x y*)
  consider (*le*) $x \le n\ y \le n$ | (*beyond*) $x > n \vee y > n$ **by** *fastforce*
  then show *?thesis*
  proof *cases*
   case *le*
   have $h'\ n\ xy = h\ n\ x\ y$ **by**(*simp add*: h'*-def le*)
   also have $\ldots \le h\ n\ x\ y + sum\ (h\ n\ x)\ \{..<y\} + sum\ (h\ n\ x)\ \{y<..n + 1\}$
    using *h-nonneg le* **by**(*auto intro*!: *sum-nonneg add-nonneg-nonneg*)
   also have $\ldots = sum\ (h\ n\ x)\ \{..y\} +\ sum\ (h\ n\ x)\ \{y<..n + 1\}$
    by(*simp add*: *sum-lessThan-conv-atMost-nat*)
   also have $\ldots = sum\ (h\ n\ x)\ \{..n+1\}$ **using** *le*
    by(*subst sum.union-disjoint*[*symmetric*])(*auto simp del*: *One-nat-def intro*!:
sum.cong)
   also have $\ldots = f'\ n\ x$ **using** *le* **by**(*simp add*: *h-f*)
   also have $\ldots = f\ x$ **using** *le* **by**(*simp add*: f'*-def*)

also have . . . = *suminf* (*zero-on f* (− {*x*}))
by(*subst suminf-finite*[**where** *N*={*x*}]) *simp-all*
also have . . . ≤ *suminf f*
by(*rule suminf-le*)(*auto simp add*: *f-nonneg summ-zf summ-f*)
also have . . . = *s* **using** *sum-f* **by**(*simp add*: *sums-unique*[*symmetric*])
finally show *?thesis* **.**
next
case *beyond*
then have *h′ n xy* = *0* **by**(*auto simp add*: *h′-def*)
also have *0* ≤ *s* **using** *summ-f* **by**(*simp add*: *sums-unique*[*OF sum-f*] *suminf-nonneg f-nonneg*)
finally show *?thesis* **.**
qed
qed
then have *bounded* (*range* (λ*n*. *h′ n x*)) **for** *x* **unfolding** *bounded-def*
by(*intro exI*[*of - 0*] *exI*[*of - s*]; *simp add*: *h′-nonneg*)
from *convergent-bounded-family*[*OF this*, *of UNIV %x*. *x*] **obtain** *k*
where *k*: *strict-mono k* **and** *conv*: ⋀*xy*. *convergent* (λ*n*. *h′* (*k n*) *xy*) **by** *auto*

define *H* :: *nat* ⇒ *nat* ⇒ *real*
where *H x y* = *lim* (λ*n*. *h′* (*k n*) (*x*, *y*)) **for** *x y*

have *H*: (λ*n*. *h′* (*k n*) (*x*, *y*)) ⟶ *H x y* **for** *x y*
unfolding *H-def* **using** *conv*[*of* (*x*, *y*)] **by**(*simp add*: *convergent-LIMSEQ-iff*)

show *thesis*
proof(*rule that*)
show *H-nonneg*: *0* ≤ *H x y* **for** *x y* **using** *H*[*of x y*] **by**(*rule LIMSEQ-le-const*)(*simp add*: *h′-nonneg*)
show (*x*, *y*) ∈ *R* **if** *0* < *H x y* **for** *x y*
proof(*rule ccontr*)
assume (*x*, *y*) ∉ *R*
hence *h′ n* (*x*, *y*) = *0* **for** *n* **using** *h-nonneg*[*of x n y*] *h-R*[*of n x y*]
by(*fastforce simp add*: *h′-def*)
hence *H x y* = *0* **using** *H*[*of x y*] **by**(*simp add*: *LIMSEQ-const-iff*)
with *that* **show** *False* **by** *simp*
qed
show *H x sums f x* **for** *x* **unfolding** *sums-iff*
proof
have *sum-H*: *sum* (*H x*) {..<*m*} ≤ *f x* **for** *m*
proof −
have *sum* (λ*y*. *h′* (*k n*) (*x*, *y*)) {..<*m*} ≤ *f x* **for** *n*
proof(*cases x* ≤ *k n*)
case *True*
from *k* **have** *n* ≤ *k n* **by**(*rule seq-suble*)
have *sum* (λ*y*. *h′* (*k n*) (*x*, *y*)) {..<*m*} = *sum* (λ*y*. *h′* (*k n*) (*x*, *y*)) {..<*min m* (*k n* + *1*)}
by(*rule sum.mono-neutral-right*)(*auto simp add*: *h′-def min-def*)
also have . . . ≤ *sum* (λ*y*. *h* (*k n*) *x y*) {..*k n* + *1*} **using** *True*

by(*intro sum-le-included*[**where** *i=id*])(*auto simp add*: *h′-def h-nonneg*)
also have … $=$ *f′* (*k n*) *x* **using** *h-f True* **by** *simp*
also have … $=$ *f x* **using** *True* **by**(*simp add*: *f′-def*)
finally show *?thesis* **.**
qed(*simp add*: *f-nonneg h′-def*)
then show *?thesis* **by** −((*rule LIMSEQ-le-const2 tendsto-sum H*)+, *simp*)
qed
with *H-nonneg* **show** *summ-H*: *summable* (*H x*) **by**(*rule summableI-nonneg-bounded*)
hence *suminf* (*H x*) $\leq$ *f x* **using** *sum-H* **by**(*rule suminf-le-const*)
moreover
have (λ*m*. *sum* (*H x*) {..<*m*} + *suminf* (λ*n*. *g* (*n* + *m*))) $\longrightarrow$ *suminf* (*H x*) + *0*
by(*rule tendsto-intros summable-LIMSEQ summ-H suminf-exist-split2 summ-g*)+
hence *f x* $\leq$ *suminf* (*H x*) + *0*
proof(*rule LIMSEQ-le-const*)
have *f x* $\leq$ *sum* (*H x*) {..<*m*} + *suminf* (λ*n*. *g* (*n* + *m*)) **for** *m*
proof −
have (λ*n*. *sum* (λ*y*. *h′* (*k n*) (*x*, *y*)) {..<*m*} + *suminf* (λ*i*. *g* (*i* + *m*))) $\longrightarrow$ *sum* (*H x*) {..<*m*} + *suminf* (λ*i*. *g* (*i* + *m*))
by(*rule tendsto-intros H*)+
moreover have $\exists$ *N*. $\forall$ *n*$\geq$*N*. *f x* $\leq$ *sum* (λ*y*. *h′* (*k n*) (*x*, *y*)) {..<*m*} + *suminf* (λ*i*. *g* (*i* + *m*))
proof(*intro exI strip*)
fix *n*
assume *max x m* $\leq$ *n*
with *seq-suble*[*OF k*, *of n*] **have** *x*: *x* $\leq$ *k n* **and** *m*: *m* $\leq$ *k n* **by** *auto*
have *f x* $=$ *f′* (*k n*) *x* **using** *x* **by**(*simp add*: *f′-def*)
also have … $=$ *sum* (*h* (*k n*) *x*) {..*k n* + *1*} **using** *x* **by**(*simp add*: *h-f*)
also have … $=$ *sum* (*h* (*k n*) *x*) {..<*m*} + *sum* (*h* (*k n*) *x*) {*m*..*k n* + *1*}
using *x m* **by**(*subst sum.union-disjoint*[*symmetric*])(*auto intro*!: *sum.cong simp del*: *One-nat-def*)
also have *sum* (*h* (*k n*) *x*) {..<*m*} $=$ *sum* (λ*y*. *h′* (*k n*) (*x*, *y*)) {..<*m*}
using *x m* **by**(*auto simp add*: *h′-def*)
also have *sum* (*h* (*k n*) *x*) {*m*..*k n* + *1*} $=$ *sum* (λ*y*. *sum* (λ*x*. *h* (*k n*) *x y*) {*x*}) {*m*..*k n* + *1*} **by** *simp*
also have … $\leq$ *sum* (λ*y*. *sum* (λ*x*. *h* (*k n*) *x y*) {..*k n* + *1*}) {*m*..*k n* + *1*} **using** *x*
by(*intro sum-mono sum-mono2*)(*auto simp add*: *h-nonneg*)
also have … $=$ *sum* (*g′* (*k n*)) {*m*..*k n* + *1*} **by**(*simp add*: *h-g del*: *One-nat-def*)
also have … $=$ *sum g* {*m*..*k n*} + *suminf* (λ*i*. *g* (*i* + (*k n* + *1*))) **using** *m* **by**(*simp add*: *g′-def*)
also have … $=$ *suminf* (λ*i*. *g* (*i* + *m*)) **using** *m*
apply(*subst* (*2*) *suminf-split-initial-segment*[**where** *k=k n* + *1* − *m*])
apply(*simp-all add*: *summable-iff-shift summ-g*)
apply(*rule sum.reindex-cong*[*OF - - refl*])
apply(*simp-all add*: *Suc-diff-le lessThan-Suc-atMost*)
apply(*safe*; *clarsimp*)
subgoal for *x* **by**(*rule image-eqI*[**where** *x=x* − *m*]) *auto*

```
          subgoal by auto
          done
        finally show f x ≤ sum (λy. h′ (k n) (x, y)) {..<m} + … by simp
      qed
      ultimately show ?thesis by(rule LIMSEQ-le-const)
    qed
    thus ∃N. ∀n≥N. f x ≤ sum (H x) {..<n} + (∑i. g (i + n)) by auto
  qed
  ultimately show suminf (H x) = f x by simp
qed
show (λx. H x y) sums g y for y unfolding sums-iff
proof
  have sum-H: sum (λx. H x y) {..<m} ≤ g y for m
  proof −
    have sum (λx. h′ (k n) (x, y)) {..<m} ≤ g y for n
    proof(cases y ≤ k n)
      case True
      from k have n ≤ k n by(rule seq-suble)
      have sum (λx. h′ (k n) (x, y)) {..<m} = sum (λx. h′ (k n) (x, y)) {..<min
m (k n + 1)}
        by(rule sum.mono-neutral-right)(auto simp add: h′-def min-def)
      also have … ≤ sum (λx. h (k n) x y) {..k n + 1} using True
        by(intro sum-le-included[where i=id])(auto simp add: h′-def h-nonneg)
      also have … = g′ (k n) y using h-g True by simp
      also have … = g y using True by(simp add: g′-def)
      finally show ?thesis .
    qed(simp add: g-nonneg h′-def)
    then show ?thesis by −((rule LIMSEQ-le-const2 tendsto-sum H)+, simp)
  qed
  with H-nonneg show summ-H: summable (λx. H x y) by(rule summableI-nonneg-bounded)
  hence suminf (λx. H x y) ≤ g y using sum-H by(rule suminf-le-const)
  moreover
  have (λm. sum (λx. H x y) {..<m} + suminf (λn. f (n + m))) ⟶ suminf
(λx. H x y) + 0
  by(rule tendsto-intros summable-LIMSEQ summ-H suminf-exist-split2 summ-f)+
  hence g y ≤ suminf (λx. H x y) + 0
  proof(rule LIMSEQ-le-const)
    have g y ≤ sum (λx. H x y) {..<m} + suminf (λn. f (n + m)) for m
    proof −
      have (λn. sum (λx. h′ (k n) (x, y)) {..<m} + suminf (λi. f (i + m)))
⟶ sum (λx. H x y) {..<m} + suminf (λi. f (i + m))
        by(rule tendsto-intros H)+
      moreover have ∃N. ∀n≥N. g y ≤ sum (λx. h′ (k n) (x, y)) {..<m} +
suminf (λi. f (i + m))
      proof(intro exI strip)
        fix n
        assume max y m ≤ n
        with seq-suble[OF k, of n] have y: y ≤ k n and m: m ≤ k n by auto
        have g y = g′ (k n) y using y by(simp add: g′-def)
```

also have $\ldots = sum\ (\lambda x.\ h\ (k\ n)\ x\ y)\ \{..k\ n + 1\}$ **using** y **by**($simp\ add$: $h\text{-}g$)
also have $\ldots = sum\ (\lambda x.\ h\ (k\ n)\ x\ y)\ \{..<m\} + sum\ (\lambda x.\ h\ (k\ n)\ x\ y)\ \{m..k\ n + 1\}$
using $y\ m$ **by**($subst\ sum.union\text{-}disjoint[symmetric]$)($auto\ intro!$: $sum.cong\ simp\ del$: $One\text{-}nat\text{-}def$)
also have $sum\ (\lambda x.\ h\ (k\ n)\ x\ y)\ \{..<m\} = sum\ (\lambda x.\ h'\ (k\ n)\ (x,\ y))\ \{..<m\}$
using $y\ m$ **by**($auto\ simp\ add$: $h'\text{-}def$)
also have $sum\ (\lambda x.\ h\ (k\ n)\ x\ y)\ \{m..k\ n + 1\} = sum\ (\lambda x.\ sum\ (\lambda y.\ h\ (k\ n)\ x\ y)\ \{y\})\ \{m..k\ n + 1\}$ **by** $simp$
also have $\ldots \leq sum\ (\lambda x.\ sum\ (\lambda y.\ h\ (k\ n)\ x\ y)\ \{..k\ n + 1\})\ \{m..k\ n + 1\}$ **using** y
by($intro\ sum\text{-}mono\ sum\text{-}mono2$)($auto\ simp\ add$: $h\text{-}nonneg$)
also have $\ldots = sum\ (f'\ (k\ n))\ \{m..k\ n + 1\}$ **by**($simp\ add$: $h\text{-}f\ del$: $One\text{-}nat\text{-}def$)
also have $\ldots = sum\ f\ \{m..k\ n\} + suminf\ (\lambda i.\ f\ (i + (k\ n + 1)))$ **using** m **by**($simp\ add$: $f'\text{-}def$)
also have $\ldots = suminf\ (\lambda i.\ f\ (i + m))$ **using** m
apply($subst\ (2)\ suminf\text{-}split\text{-}initial\text{-}segment$[**where** k=$k\ n + 1 - m$])
apply($simp\text{-}all\ add$: $summable\text{-}iff\text{-}shift\ summ\text{-}f$)
apply($rule\ sum.reindex\text{-}cong[OF\ \text{-}\ \text{-}\ refl]$)
apply($simp\text{-}all\ add$: $Suc\text{-}diff\text{-}le\ lessThan\text{-}Suc\text{-}atMost$)
apply($safe$; $clarsimp$)
subgoal for x **by**($rule\ image\text{-}eqI$[**where** x=$x - m$]) $auto$
subgoal by $auto$
done
finally show $g\ y \leq sum\ (\lambda x.\ h'\ (k\ n)\ (x,\ y))\ \{..<m\} + \ldots$ **by** $simp$
qed
ultimately show $?thesis$ **by**($rule\ LIMSEQ\text{-}le\text{-}const$)
qed
thus $\exists N.\ \forall n{\geq}N.\ g\ y \leq sum\ (\lambda x.\ H\ x\ y)\ \{..<n\} + (\sum i.\ f\ (i + n))$ **by** $auto$
qed
ultimately show $suminf\ (\lambda x.\ H\ x\ y) = g\ y$ **by** $simp$
qed
qed
qed

lemma $bounded\text{-}matrix\text{-}for\text{-}marginals\text{-}ennreal$:
assumes $sum\text{-}eq$: $(\sum^+ x{\in}A.\ f\ x) = (\sum^+ y{\in}B.\ g\ y)$
and $finite$: $(\sum^+ x{\in}B.\ g\ x) \neq \top$
and le: $\bigwedge X.\ X \subseteq A \Longrightarrow (\sum^+ x{\in}X.\ f\ x) \leq (\sum^+ y{\in}R\ ``\ X.\ g\ y)$
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 $\bigwedge x\ y.\ h\ x\ y \neq \top$
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 $-$

have *fin-g* [*simp*]: $g\ y \neq \top$ **if** $y \in B$ **for** y **using** *finite*
by(*rule neq-top-trans*)(*rule nn-integral-ge-point*[*OF that*])
have *fin-f* [*simp*]: $f\,x \neq \top$ **if** $x \in A$ **for** x **using** *finite* **unfolding** *sum-eq*[*symmetric*]
by(*rule neq-top-trans*)(*rule nn-integral-ge-point*[*OF that*])

define f' **where** $f'\ x = ($*if* $x \in$ *to-nat-on* A ‘ A *then enn2real* (f (*from-nat-into* $A\ x$)) *else* 0) **for** x
define g' **where** $g'\ y = ($*if* $y \in$ *to-nat-on* B ‘ B *then enn2real* (g (*from-nat-into* $B\ y$)) *else* 0) **for** y
define s **where** $s =$ *enn2real* ($\sum^+ x{\in}B.\ g\ x$)
define R' **where** $R' =$ *map-prod* (*to-nat-on* A) (*to-nat-on* B) ‘ R

have *f′-nonneg*: $f'\ x \geq 0$ **for** x **by**(*simp add*: *f′-def*)
have *g′-nonneg*: $g'\ y \geq 0$ **for** y **by**(*simp add*: *g′-def*)

have ($\sum^+ x.\ f'\ x$) = ($\sum^+ x{\in}$*to-nat-on* A ‘ $A.\ f'\ x$)
by(*auto simp add*: *nn-integral-count-space-indicator f′-def intro*!: *nn-integral-cong*)
also have ... = ($\sum^+ x{\in}A.\ f\ x$)
by(*subst nn-integral-count-space-reindex*)(*auto simp add*: *inj-on-to-nat-on f′-def ennreal-enn2real-if intro*!: *nn-integral-cong*)
finally have *sum-f′*: ($\sum^+ x.\ f'\ x$) = ($\sum^+ x{\in}A.\ f\ x$) **.**

have ($\sum^+ y.\ g'\ y$) = ($\sum^+ y{\in}$*to-nat-on* B ‘ $B.\ g'\ y$)
by(*auto simp add*: *nn-integral-count-space-indicator g′-def intro*!: *nn-integral-cong*)
also have ... = ($\sum^+ y{\in}B.\ g\ y$)
by(*subst nn-integral-count-space-reindex*)(*auto simp add*: *inj-on-to-nat-on g′-def ennreal-enn2real-if intro*!: *nn-integral-cong*)
finally have *sum-g′*: ($\sum^+ y.\ g'\ y$) = ($\sum^+ y{\in}B.\ g\ y$) **.**

have *summ-f′*: *summable* f'
proof(*rule summableI-nonneg-bounded*)
show *sum* f' $\{..{<}n\} \leq$ *enn2real* ($\sum^+ x.\ f'\ x$) **for** n
proof −
have *sum* f' $\{..{<}n\}$ = *enn2real* ($\sum^+ x{\in}\{..{<}n\}.\ f'\ x$)
by(*simp add*: *nn-integral-count-space-finite f′-nonneg sum-nonneg*)
also have *enn2real* ($\sum^+ x{\in}\{..{<}n\}.\ f'\ x$) $\leq$ *enn2real* ($\sum^+ x.\ f'\ x$) **using** *finite sum-eq*[*symmetric*]
by(*auto simp add*: *nn-integral-count-space-indicator sum-f′*[*symmetric*] *less-top intro*!: *nn-integral-mono enn2real-mono split*: *split-indicator*)
finally show *?thesis* **.**
qed
qed(*rule f′-nonneg*)
have *suminf-f′*: *suminf* f' = *enn2real* ($\sum^+ y.\ f'\ y$)
by(*simp add*: *nn-integral-count-space-nat suminf-ennreal2* [*OF f′-nonneg summ-f′*] *suminf-nonneg*[*OF summ-f′*] *f′-nonneg*)
with *summ-f′ sum-f′ sum-eq* **have** *sums-f*: f' *sums* s **by**(*simp add*: *s-def sums-iff*)
moreover
have *summ-g′*: *summable* g'
proof(*rule summableI-nonneg-bounded*)

show *sum g′* {..<*n*} ≤ *enn2real* (∑⁺ *y*. *g′ y*) **for** *n*
proof −
have *sum g′* {..<*n*} = *enn2real* (∑⁺ *y*∈{..<*n*}. *g′ y*)
by(*simp add*: *nn-integral-count-space-finite g′-nonneg sum-nonneg*)
also have *enn2real* (∑⁺ *y*∈{..<*n*}. *g′ y*) ≤ *enn2real* (∑⁺ *y*. *g′ y*) **using** *finite*
by(*auto simp add*: *nn-integral-count-space-indicator sum-g′*[*symmetric*] *less-top intro*!: *nn-integral-mono enn2real-mono split*: *split-indicator*)
finally show *?thesis* **.**
qed
qed(*rule g′-nonneg*)
have *suminf-g′*: *suminf g′* = *enn2real* (∑⁺ *y*. *g′ y*)
by(*simp add*: *nn-integral-count-space-nat suminf-ennreal2*[*OF g′-nonneg summ-g′*] *suminf-nonneg*[*OF summ-g′*] *g′-nonneg*)
with *summ-g′ sum-g′* **have** *sums-g*: *g′ sums s* **by**(*simp add*: *s-def sums-iff*)
moreover note *f′-nonneg g′-nonneg*
moreover have *suminf* (*zero-on f′* (− *X*)) ≤ *suminf* (*zero-on g′* (− *R′* ‘‘ *X*)) **for** *X*
proof −
define *X′* **where** *X′* = *from-nat-into A* ‘ (*X* ∩ *to-nat-on A* ‘ *A*)
have *X′*: *to-nat-on A* ‘ *X′* = *X* ∩ (*to-nat-on A* ‘ *A*)
by(*auto 4 3 simp add*: *X′-def intro*: *rev-image-eqI*)

have *ennreal* (*suminf* (*zero-on f′* (− *X*))) = *suminf* (*zero-on* (λ*x*. *ennreal* (*f′ x*)) (− *X*))
by(*simp add*: *suminf-ennreal2 zero-on-nonneg f′-nonneg summable-zero-on-nonneg summ-f′*)
also have . . . = (∑⁺ *x*∈*X*. *f′ x*)
by(*auto simp add*: *nn-integral-count-space-nat*[*symmetric*] *nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have . . . = (∑⁺ *x*∈*to-nat-on A* ‘ *X′*. *f′ x*) **using** *X′*
by(*auto simp add*: *nn-integral-count-space-indicator f′-def intro*!: *nn-integral-cong split*: *split-indicator*)
also have . . . = (∑⁺ *x* ∈ *X′*. *f x*)
by(*subst nn-integral-count-space-reindex*)(*auto simp add*: *X′-def inj-on-def f′-def ennreal-enn2real-if intro*!: *nn-integral-cong*)
also have . . . ≤ (∑⁺ *y* ∈ *R* ‘‘ *X′*. *g y*) **by**(*rule le*)(*auto simp add*: *X′-def*)
also have . . . = (∑⁺ *y* ∈ *to-nat-on B* ‘ (*R* ‘‘ *X′*). *g′ y*) **using** *R fin-g*
by(*subst nn-integral-count-space-reindex*)(*auto 4 3 simp add*: *X′-def inj-on-def g′-def ennreal-enn2real-if simp del*: *fin-g intro*!: *nn-integral-cong from-nat-into dest*: *to-nat-on-inj*[*THEN iffD1*, *rotated −1*])
also have *to-nat-on B* ‘ (*R* ‘‘ *X′*) = *R′* ‘‘ *X* **using** *R*
by(*auto 4 4 simp add*: *X′-def R′-def Image-iff intro*: *rev-image-eqI rev-bexI intro*!: *imageI*)
also have (∑⁺ *y*∈. . . . *g′ y*) = *suminf* (*zero-on* (λ*y*. *ennreal* (*g′ y*)) (− . . .))
by(*auto simp add*: *nn-integral-count-space-nat*[*symmetric*] *nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have . . . = *ennreal* (*suminf* (*zero-on g′* (− *R′* ‘‘ *X*)))
by(*simp add*: *suminf-ennreal2 zero-on-nonneg g′-nonneg summable-zero-on-nonneg*

summm-g′)
  finally show *?thesis*
  by(*simp add*: *suminf-nonneg summable-zero-on-nonneg*[*OF summ-g*′ *g*′*-nonneg*] *zero-on-nonneg g*′*-nonneg*)
 qed
 ultimately obtain h' **where** h'*-nonneg*: $\bigwedge x\ y.\ 0 \le h'\ x\ y$
  and *dom-h*′: $\bigwedge x\ y.\ 0 < h'\ x\ y \Longrightarrow (x,\ y) \in R'$
  and h'*-f*: $\bigwedge x.\ h'\ x$ *sums* $f'\ x$
  and h'*-g*: $\bigwedge y.\ (\lambda x.\ h'\ x\ y)$ *sums* $g'\ y$
  by(*rule bounded-matrix-for-marginals-nat*) *blast*

 define h **where** $h\ x\ y =$ *ennreal* (*if* $x \in A \wedge y \in B$ *then* h' (*to-nat-on* $A\ x$) (*to-nat-on* $B\ y$) *else* 0) **for** $x\ y$
 show *?thesis*
 proof
  show $(x,\ y) \in R$ **if** $0 < h\ x\ y$ **for** $x\ y$
   using *that dom-h*′[*of to-nat-on A x to-nat-on B y*] R
   by(*auto simp add*: *h-def R*′*-def dest*: *to-nat-on-inj*[*THEN iffD1*, *rotated* -1] *split*: *if-split-asm*)
  show $h\ x\ y \neq \top$ **for** $x\ y$ **by**(*simp add*: *h-def*)

  fix x
  assume x: $x \in A$
  have $(\sum^+ y{\in}B.\ h\ x\ y) = (\sum^+ y{\in}$*to-nat-on* B ‘ $B.\ h'$ (*to-nat-on* $A\ x$) $y)$
  by(*subst nn-integral-count-space-reindex*)(*auto simp add*: *inj-on-to-nat-on h-def x intro*!: *nn-integral-cong*)
  also have … $= (\sum^+ y.\ h'$ (*to-nat-on* $A\ x$) $y)$ **using** *dom-h*′[*of to-nat-on A x*] h'*-nonneg* R
  by(*fastforce intro*!: *nn-integral-cong intro*: *rev-image-eqI simp add*: *nn-integral-count-space-indicator R*′*-def less-le split*: *split-indicator*)
  also have … = *ennreal* (*suminf* (h' (*to-nat-on* $A\ x$)))
  by(*simp add*: *nn-integral-count-space-nat suminf-ennreal-eq*[*OF - h*′*-f*] h'*-nonneg*)

  also have … = *ennreal* (f' (*to-nat-on* $A\ x$)) **using** h'*-f*[*of to-nat-on A x*] **by**(*simp add*: *sums-iff*)
  also have … $= f\ x$ **using** x **by**(*simp add*: *f*′*-def ennreal-enn2real-if*)
  finally show $(\sum^+ y{\in}B.\ h\ x\ y) = f\ x$ **.**
 next
  fix y
  assume y: $y \in B$
  have $(\sum^+ x{\in}A.\ h\ x\ y) = (\sum^+ x{\in}$*to-nat-on* A ‘ $A.\ h'\ x$ (*to-nat-on* $B\ y$))
  by(*subst nn-integral-count-space-reindex*)(*auto simp add*: *inj-on-to-nat-on h-def y intro*!: *nn-integral-cong*)
  also have … $= (\sum^+ x.\ h'\ x$ (*to-nat-on* $B\ y$)) **using** *dom-h*′[*of - to-nat-on B y*] h'*-nonneg* R
  by(*fastforce intro*!: *nn-integral-cong intro*: *rev-image-eqI simp add*: *nn-integral-count-space-indicator R*′*-def less-le split*: *split-indicator*)
  also have … = *ennreal* (*suminf* ($\lambda x.\ h'\ x$ (*to-nat-on* $B\ y$)))
  by(*simp add*: *nn-integral-count-space-nat suminf-ennreal-eq*[*OF - h*′*-g*] h'*-nonneg*)

also have ... $=$ *ennreal* (g' (*to-nat-on B y*)) **using** h'-*g*[*of to-nat-on B y*] **by**(*simp add*: *sums-iff*)
also have ... $=$ *g y* **using** *y* **by**(*simp add*: g'-*def ennreal-enn2real-if*)
finally show ($\sum^{+}$ $x{\in}A$. *h x y*) $=$ *g y* **.**
qed
qed

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* (‹**E**ı›)
where $\mathbf{E}_G \equiv \{(x, y).\ \mathit{edge}\ G\ x\ y\}$

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

definition *incoming* :: ($'v$, $'more$) *graph-scheme* $\Rightarrow 'v \Rightarrow 'v$ *set* (‹**IN**ı›)
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* $\vee$ *Rangep* (*edge G*) *x*

lemma *vertexI*:
shows *vertexI1*: *edge* Γ *x y* $\Longrightarrow$ *vertex* Γ *x*
and *vertexI2*: *edge* Γ *x y* $\Longrightarrow$ *vertex* Γ *y*
by(*auto simp add*: *vertex-def*)

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

lemma **V**-*def*: $\mathbf{V}_G = \mathit{fst}$ ‘ $\mathbf{E}_G \cup \mathit{snd}$ ‘ $\mathbf{E}_G$

by(*auto 4 3 simp add*: *vertex-def intro*: *rev-image-eqI prod.expand*)

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

abbreviation *path* :: (*'v*, *'more*) *graph-scheme* ⇒ *'v* ⇒ *'v path* ⇒ *'v* ⇒ *bool*
where *path G* ≡ *rtrancl-path* (*edge G*)

inductive *cycle* :: (*'v*, *'more*) *graph-scheme* ⇒ *'v path* ⇒ *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: ⟦ *path G v p v*; *p* ≠ []; *distinct p* ⟧ ⟹ *cycle G p*

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

abbreviation *cycles* :: (*'v*, *'more*) *graph-scheme* ⇒ *'v path set*
where *cycles G* ≡ *Collect* (*cycle G*)

lemma *countable-cycles* [*simp*]:
  assumes *countable* ($\mathbf{V}_G$)
  shows *countable* (*cycles G*)
proof −
  have *cycles G* ⊆ *lists* $\mathbf{V}_G$
    by(*auto elim*!: *cycle.cases dest*: *rtrancl-path-Range-end rtrancl-path-Range simp add*: *vertex-def*)
  thus *?thesis* **by**(*rule countable-subset*)(*simp add*: *assms*)
qed

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

lemma *cycle-edges-not-Nil*: *cycle G p* ⟹ *cycle-edges p* ≠ []
by(*auto simp add*: *cycle-edges-def cycle.simps neq-Nil-conv zip-Cons1 split*: *list.split*)

lemma *distinct-cycle-edges*:
  cycle G p ⟹ *distinct* (*cycle-edges p*)
by(*erule cycle.cases*)(*simp add*: *cycle-edges-def distinct-zipI2*)

lemma *cycle-enter-leave-same*:
  assumes *cycle G p*
  shows *card* (*set* [(*x'*, *y*) ← *cycle-edges p*. *x'* = *x*]) = *card* (*set* [(*x'*, *y*) ← *cycle-edges p*. *y* = *x*])
  (**is** *?lhs* = *?rhs*)
using *assms*
proof *cases*
  case (*cycle v*)
  from *distinct-cycle-edges*[*OF assms*]
  have *?lhs* = *length* [*x'* ← *map fst* (*cycle-edges p*). *x'* = *x*]

by(*subst distinct-card*; *simp add*: *filter-map o-def split-def*)
also have ... = (*if* $x \in$ *set p then 1 else 0*) **using** *cycle*
by(*auto simp add*: *cycle-edges-def filter-empty-conv length-filter-conv-card card-eq-1-iff in-set-conv-nth dest*: *nth-eq-iff-index-eq*)
also have ... = *length* [$y \leftarrow$ *map snd* (*cycle-edges p*). $y = x$] **using** *cycle*
apply(*auto simp add*: *cycle-edges-def filter-empty-conv Suc-length-conv intro*!: *exI*[**where** x=x])
apply(*drule split-list-first*)
apply(*auto dest*: *split-list-first simp add*: *append-eq-Cons-conv rotate1-append filter-empty-conv split*: *if-split-asm dest*: *in-set-tlD*)
done
also have ... = *?rhs* **using** *distinct-cycle-edges*[*OF assms*]
by(*subst distinct-card*; *simp add*: *filter-map o-def split-def*)
finally show *?thesis* **.**
qed

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*)
using *assms*
by(*cases*)(*auto 4 3 simp add*: *cycle-edges-def cong*: *conj-cong split*: *if-split-asm intro*: *set-zip-rightI dest*: *set-zip-leftD*)

lemma *cycle-edges-edges*:
assumes *cycle G p*
shows *set* (*cycle-edges p*) $\subseteq \mathbf{E}_G$
proof
fix x
assume $x \in$ *set* (*cycle-edges p*)
then obtain i **where** x: $x = (p \; ! \; i,$ *rotate1* $p \; ! \; i)$ **and** i: $i <$ *length p*
by(*auto simp add*: *cycle-edges-def set-zip*)
from *assms* **obtain** v **where** p: *path G v p v* **and** $p \neq []$ **and** *distinct p* **by** *cases*
let *?i* = *Suc i mod length p*
have *?i* < *length p* **by** (*simp add*: ‹$p \neq []$›)
note *rtrancl-path-nth*[*OF p this*]
also have $(v \# p) \; ! \;$ *?i* $= p \; ! \; i$
proof(*cases Suc i* < *length p*)
case *True* **thus** *?thesis* **by** *simp*
next
case *False*
with i **have** *Suc i* = *length p* **by** *simp*
moreover from p ‹$p \neq []$› **have** *last p* = v **by**(*rule rtrancl-path-last*)
ultimately show *?thesis* **using** ‹$p \neq []$› **by**(*simp add*: *last-conv-nth*)(*metis diff-Suc-Suc diff-zero*)
qed
also have $p \; ! \;$ *?i* = *rotate1* $p \; ! \; i$ **using** i **by**(*simp add*: *nth-rotate1*)
finally show $x \in \mathbf{E}_G$ **by**(*simp add*: x)
qed

5 Network and Flow

record *'v network* = *'v graph* +
  capacity :: *'v edge* ⇒ *ennreal*
  source :: *'v*
  sink :: *'v*

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

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

lemma *support-flow-conv*: *support-flow f* = $\{e.\ f\ e > 0\}$
by(*auto simp add*: *support-flow.simps*)

lemma *not-in-support-flowD*: $x \notin$ *support-flow f* $\Longrightarrow f\ x = 0$
by(*simp add*: *support-flow-conv*)

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

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

lemma *d-OUT-mono*: $(\bigwedge y.\ f\ (x,\ y) \le g\ (x,\ y)) \Longrightarrow$ *d-OUT f x* ≤ *d-OUT g x*
by(*auto simp add*: *d-OUT-def le-fun-def intro*: *nn-integral-mono*)

lemma *d-IN-mono*: $(\bigwedge x.\ f\ (x,\ y) \le g\ (x,\ y)) \Longrightarrow$ *d-IN f y* ≤ *d-IN g y*
by(*auto simp add*: *d-IN-def le-fun-def intro*: *nn-integral-mono*)

lemma *d-OUT-0* [*simp*]: *d-OUT* (λ-. *0*) *x* = *0*
by(*simp add*: *d-OUT-def*)

lemma *d-IN-0* [*simp*]: *d-IN* (λ-. *0*) *x* = *0*
by(*simp add*: *d-IN-def*)

lemma *d-OUT-add*: *d-OUT* (λ*e*. *f e* + *g e*) *x* = *d-OUT f x* + *d-OUT g x*
unfolding *d-OUT-def* **by**(*simp add*: *nn-integral-add*)

lemma *d-IN-add*: *d-IN* (λ*e*. *f e* + *g e*) *x* = *d-IN f x* + *d-IN g x*
unfolding *d-IN-def* **by**(*simp add*: *nn-integral-add*)

lemma *d-OUT-cmult*: *d-OUT* (λ*e*. *c* ∗ *f e*) *x* = *c* ∗ *d-OUT f x*
by(*simp add*: *d-OUT-def nn-integral-cmult*)

lemma *d-IN-cmult*: *d-IN* (λ*e*. *c* ∗ *f e*) *x* = *c* ∗ *d-IN f x*
by(*simp add*: *d-IN-def nn-integral-cmult*)

lemma *d-OUT-ge-point*: $f\ (x,\ y) \leq d\text{-}OUT\ f\ x$
by(*auto simp add*: *d-OUT-def intro*!: *nn-integral-ge-point*)

lemma *d-IN-ge-point*: $f\ (y,\ x) \leq d\text{-}IN\ f\ x$
by(*auto simp add*: *d-IN-def intro*!: *nn-integral-ge-point*)

lemma *d-OUT-monotone-convergence-SUP*:
  assumes *incseq* $(\lambda n\ y.\ f\ n\ (x,\ y))$
  shows $d\text{-}OUT\ (\lambda e.\ SUP\ n.\ f\ n\ e)\ x = (SUP\ n.\ d\text{-}OUT\ (f\ n)\ x)$
unfolding *d-OUT-def* **by**(*rule nn-integral-monotone-convergence-SUP*[*OF assms*]) *simp*

lemma *d-IN-monotone-convergence-SUP*:
  assumes *incseq* $(\lambda n\ x.\ f\ n\ (x,\ y))$
  shows $d\text{-}IN\ (\lambda e.\ SUP\ n.\ f\ n\ e)\ y = (SUP\ n.\ d\text{-}IN\ (f\ n)\ y)$
unfolding *d-IN-def* **by**(*rule nn-integral-monotone-convergence-SUP*[*OF assms*]) *simp*

lemma *d-OUT-diff*:
  assumes $\bigwedge y.\ g\ (x,\ y) \leq f\ (x,\ y)\ d\text{-}OUT\ g\ x \neq \top$
  shows $d\text{-}OUT\ (\lambda e.\ f\ e - g\ e)\ x = d\text{-}OUT\ f\ x - d\text{-}OUT\ g\ x$
using *assms* **by**(*simp add*: *nn-integral-diff d-OUT-def*)

lemma *d-IN-diff*:
  assumes $\bigwedge x.\ g\ (x,\ y) \leq f\ (x,\ y)\ d\text{-}IN\ g\ y \neq \top$
  shows $d\text{-}IN\ (\lambda e.\ f\ e - g\ e)\ y = d\text{-}IN\ f\ y - d\text{-}IN\ g\ y$
using *assms* **by**(*simp add*: *nn-integral-diff d-IN-def*)

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*: $(\bigwedge x.\ (x,\ y) \notin \mathbf{E} \Longrightarrow g\ (x,\ y) = 0) \Longrightarrow d\text{-}IN\ g\ y = (\sum^{+}\ x{\in}\mathbf{IN}\ y.\ g\ (x,\ y))$
unfolding *d-OUT-def d-IN-def*
by(*fastforce simp add*: *max-def d-OUT-def d-IN-def nn-integral-count-space-indicator outgoing-def incoming-def intro*!: *nn-integral-cong split*: *split-indicator*)+

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*: $d\text{-}IN\ g\ y = (\sum^{+}\ x{\in}\{x.\ (x,\ y) \in support\text{-}flow\ g\}.\ g\ (x,\ y))$
unfolding *d-OUT-def d-IN-def*
by(*auto simp add*: *max-def d-OUT-def d-IN-def nn-integral-count-space-indicator outgoing-def incoming-def support-flow.simps intro*!: *nn-integral-cong split*: *split-indicator*)+

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* ⇒ *ennreal*) ⇒ *'v set*
  for *f*
where *SINK*: *d-OUT f x* = *0* ⟹ *x* ∈ *SINK f*

lemma *SINK-mono*:
  assumes ⋀*e*. *f e* ≤ *g e*
  shows *SINK g* ⊆ *SINK f*
proof(*rule subsetI*; *erule SINK.cases*; *hypsubst*)
  fix *x*
  assume *d-OUT g x* = *0*
  moreover have *d-OUT f x* ≤ *d-OUT g x* **using** *assms* **by**(*rule d-OUT-mono*)
  ultimately have *d-OUT f x* = *0* **by** *simp*
  thus *x* ∈ *SINK f* **..**
qed

lemma *SINK-mono'*: *f* ≤ *g* ⟹ *SINK g* ⊆ *SINK f*
by(*rule SINK-mono*)(*rule le-funD*)

lemma *support-flow-Sup*: *support-flow* (*Sup Y*) = (⋃*f*∈*Y*. *support-flow f*)
by(*auto simp add*: *support-flow-conv less-SUP-iff*)

lemma
  assumes *chain*: *Complete-Partial-Order.chain* (≤) *Y*
  and *Y*: *Y* ≠ {}
  and *countable*: *countable* (*support-flow* (*Sup Y*))
  shows *d-OUT-Sup*: *d-OUT* (*Sup Y*) *x* = (*SUP f*∈*Y*. *d-OUT f x*) (**is** *?OUT x* **is** *?lhs1 x* = *?rhs1 x*)
  and *d-IN-Sup*: *d-IN* (*Sup Y*) *y* = (*SUP f*∈*Y*. *d-IN f y*) (**is** *?IN* **is** *?lhs2* = *?rhs2*)
  and *SINK-Sup*: *SINK* (*Sup Y*) = (⋂*f*∈*Y*. *SINK f*) (**is** *?SINK*)
proof −
  have *chain'*: *Complete-Partial-Order.chain* (≤) ((λ*f y*. *f* (*x*, *y*)) ‘ *Y*) **for** *x* **using** *chain*
    by(*rule chain-imageI*)(*simp add*: *le-fun-def*)
  have *countable'*: *countable* {*y*. (*x*, *y*) ∈ *support-flow* (*Sup Y*)} **for** *x*
    using - *countable*[*THEN countable-image*[**where** *f*=*snd*]]
    by(*rule countable-subset*)(*auto intro*: *prod.expand rev-image-eqI*)
  { **fix** *x*
    have *?lhs1 x* = ($\sum^+$ *y*∈{*y*. (*x*, *y*) ∈ *support-flow* (*Sup Y*)}. *SUP f*∈*Y*. *f* (*x*, *y*))
      by(*subst d-OUT-alt-def2*; *simp*)
    also have ... = (*SUP f*∈*Y*. $\sum^+$ *y*∈{*y*. (*x*, *y*) ∈ *support-flow* (*Sup Y*)}. *f* (*x*, *y*)) **using** *Y*
        by(*rule nn-integral-monotone-convergence-SUP-countable*)(*auto simp add*: *chain'* *intro*: *countable'*)
    also have ... = *?rhs1 x* **unfolding** *d-OUT-alt-def2*
     by(*auto 4 3 simp add*: *support-flow-Sup max-def nn-integral-count-space-indicator intro*!: *nn-integral-cong SUP-cong split*: *split-indicator dest*: *not-in-support-flowD*)
    finally show *?OUT x* **.** }

note *out* = *this*

have *chain''*: *Complete-Partial-Order.chain* (≤) ((λ*f x*. *f* (*x*, *y*)) ' *Y*) **for** *y* **using** *chain*
by(*rule chain-imageI*)(*simp add*: *le-fun-def*)
have *countable''*: *countable* {*x*. (*x*, *y*) ∈ *support-flow* (*Sup Y*)} **for** *y*
using - *countable*[*THEN countable-image*[**where** *f*=*fst*]]
by(*rule countable-subset*)(*auto intro*: *prod.expand rev-image-eqI*)
have *?lhs2* = ($\sum^+$ *x*∈{*x*. (*x*, *y*) ∈ *support-flow* (*Sup Y*)}. *SUP f*∈*Y*. *f* (*x*, *y*))
by(*subst d-IN-alt-def2*; *simp*)
also have ... = (*SUP f*∈*Y*. $\sum^+$ *x*∈{*x*. (*x*, *y*) ∈ *support-flow* (*Sup Y*)}. *f* (*x*, *y*)) **using** *Y*
by(*rule nn-integral-monotone-convergence-SUP-countable*)(*simp-all add*: *chain'' countable''*)
also have ... = *?rhs2* **unfolding** *d-IN-alt-def2*
by(*auto 4 3 simp add*: *support-flow-Sup max-def nn-integral-count-space-indicator intro*!: *nn-integral-cong SUP-cong split*: *split-indicator dest*: *not-in-support-flowD*)
finally show *?IN* **.**

show *?SINK* **by**(*rule set-eqI*)(*simp add*: *SINK.simps out Y bot-ennreal*[*symmetric*])
qed

lemma
assumes *chain*: *Complete-Partial-Order.chain* (≤) *Y*
and *Y*: *Y* ≠ {}
and *countable*: *countable* (*support-flow f*)
and *bounded*: ⋀*g e*. *g* ∈ *Y* ⟹ *g e* ≤ *f e*
shows *d-OUT-Inf*: *d-OUT f x* ≠ *top* ⟹ *d-OUT* (*Inf Y*) *x* = (*INF g*∈*Y*. *d-OUT g x*) (**is** - ⟹ *?OUT* **is** - ⟹ *?lhs1* = *?rhs1*)
and *d-IN-Inf*: *d-IN f x* ≠ *top* ⟹ *d-IN* (*Inf Y*) *x* = (*INF g*∈*Y*. *d-IN g x*) (**is** - ⟹ *?IN* **is** - ⟹ *?lhs2* = *?rhs2*)
proof −

We take a detour here via suprema because we have more theorems about $integral^N$ with suprema than with infinma.

from *Y* **obtain** *g0* **where** *g0*: *g0* ∈ *Y* **by** *auto*
have *g0-le-f*: *g0 e* ≤ *f e* **for** *e* **by**(*rule bounded*[*OF g0*])

have *support-flow* (*SUP g*∈*Y*. (λ*e*. *f e* − *g e*)) ⊆ *support-flow f*
by(*clarsimp simp add*: *support-flow.simps less-SUP-iff elim*!: *less-le-trans intro*!: *diff-le-self-ennreal*)
then have *countable'*: *countable* (*support-flow* (*SUP g*∈*Y*. (λ*e*. *f e* − *g e*))) **by**(*rule countable-subset*)(*rule countable*)

have *Complete-Partial-Order.chain* (≥) *Y* **using** *chain* **by**(*simp add*: *chain-dual*)
hence *chain'*: *Complete-Partial-Order.chain* (≤) ((λ*g e*. *f e* − *g e*) ' *Y*)
by(*rule chain-imageI*)(*auto simp add*: *le-fun-def intro*: *ennreal-minus-mono*)

{ **assume** *finite*: *d-OUT f x* ≠ *top*

have *finite'* [*simp*]: $f\ (x,\ y) \neq \top$ **for** y **using** *finite*
by(*rule neq-top-trans*) (*rule d-OUT-ge-point*)

have *finite'-g*: $g\ (x,\ y) \neq \top$ **if** $g \in Y$ **for** $g\ y$ **using** *finite'*[*of y*]
by(*rule neq-top-trans*)(*rule bounded*[*OF that*])

have *finite1*: $(\sum^+\ y.\ f\ (x,\ y) - (INF\ g{\in}Y.\ g\ (x,\ y))) \neq top$
using *finite* **by**(*rule neq-top-trans*)(*auto simp add*: *d-OUT-def intro*!: *nn-integral-mono*)
have *finite2*: *d-OUT g x* $\neq$ *top* **if** $g \in Y$ **for** g **using** *finite*
by(*rule neq-top-trans*)(*auto intro*: *d-OUT-mono bounded*[*OF that*])

have *bounded1*: $(\prod g{\in}Y.\ d\text{-}OUT\ g\ x) \leq d\text{-}OUT\ f\ x$
using Y **by** (*blast intro*: *INF-lower2 d-OUT-mono bounded*)

have $?lhs1 = (\sum^+\ y.\ INF\ g{\in}Y.\ g\ (x,\ y))$ **by**(*simp add*: *d-OUT-def*)
also have $\ldots = d\text{-}OUT\ f\ x - (\sum^+\ y.\ f\ (x,\ y) - (INF\ g{\in}Y.\ g\ (x,\ y)))$ **unfolding** *d-OUT-def*
using *finite1 g0-le-f*
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space intro*!: *diff-le-self-ennreal INF-lower2*[*OF g0*] *nn-integral-cong diff-diff-ennreal*[*symmetric*])
done
also have $(\sum^+\ y.\ f\ (x,\ y) - (INF\ g{\in}Y.\ g\ (x,\ y))) = d\text{-}OUT\ (\lambda e.\ SUP\ g{\in}Y.\ f\ e - g\ e)\ x$
unfolding *d-OUT-def* **by**(*subst SUP-const-minus-ennreal*)(*simp-all add*: Y)
also have $\ldots = (SUP\ h{\in}(\lambda g\ e.\ f\ e - g\ e)\ `\ Y.\ d\text{-}OUT\ h\ x)$ **using** *countable' chain' Y*
by(*subst d-OUT-Sup*[*symmetric*])(*simp-all add*: *SUP-apply*[*abs-def*])
also have $\ldots = (SUP\ g{\in}Y.\ d\text{-}OUT\ (\lambda e.\ f\ e - g\ e)\ x)$ **unfolding** *image-image* **..**
also have $\ldots = (SUP\ g{\in}Y.\ d\text{-}OUT\ f\ x - d\text{-}OUT\ g\ x)$
by(*rule SUP-cong*[*OF refl*] *d-OUT-diff*)+(*auto intro*: *bounded simp add*: *finite2*)
also have $\ldots = d\text{-}OUT\ f\ x - ?rhs1$ **by**(*subst SUP-const-minus-ennreal*)(*simp-all add*: Y)
also have $d\text{-}OUT\ f\ x - \ldots = ?rhs1$
using Y **by**(*subst diff-diff-ennreal*)(*simp-all add*: *bounded1 finite*)
finally show *?OUT* **.**
next
assume *finite*: *d-IN f x* $\neq$ *top*
have *finite'* [*simp*]: $f\ (y,\ x) \neq \top$ **for** y **using** *finite*
by(*rule neq-top-trans*) (*rule d-IN-ge-point*)

have *finite'-g*: $g\ (y,\ x) \neq \top$ **if** $g \in Y$ **for** $g\ y$ **using** *finite'*[*of y*]
by(*rule neq-top-trans*)(*rule bounded*[*OF that*])

have *finite1*: $(\sum^+\ y.\ f\ (y,\ x) - (INF\ g{\in}Y.\ g\ (y,\ x))) \neq top$
using *finite* **by**(*rule neq-top-trans*)(*auto simp add*: *d-IN-def diff-le-self-ennreal intro*!: *nn-integral-mono*)

have *finite2*: *d-IN g x* $\neq$ *top* **if** $g \in Y$ **for** *g* **using** *finite*
by(*rule neq-top-trans*)(*auto intro*: *d-IN-mono bounded*[*OF that*])

have *bounded1*: ($\prod g{\in}Y$. *d-IN g x*) $\leq$ *d-IN f x*
using *Y* **by** (*blast intro*: *INF-lower2 d-IN-mono bounded*)

have *?lhs2* = ($\sum^+$ *y*. *INF* $g{\in}Y$. *g* (*y*, *x*)) **by**(*simp add*: *d-IN-def*)
also have ... = *d-IN f x* − ($\sum^+$ *y*. *f* (*y*, *x*) − (*INF* $g{\in}Y$. *g* (*y*, *x*))) **unfolding** *d-IN-def*
using *finite1 g0-le-f*
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space intro*!: *diff-le-self-ennreal INF-lower2*[*OF g0*] *nn-integral-cong diff-diff-ennreal*[*symmetric*])
done
also have ($\sum^+$ *y*. *f* (*y*, *x*) − (*INF* $g{\in}Y$. *g* (*y*, *x*))) = *d-IN* (λ*e*. *SUP* $g{\in}Y$. *f e* − *g e*) *x*
unfolding *d-IN-def* **by**(*subst SUP-const-minus-ennreal*)(*simp-all add*: *Y*)
also have ... = (*SUP* $h{\in}$(λ*g e*. *f e* − *g e*) ‘ *Y*. *d-IN h x*) **using** *countable′ chain′ Y*
by(*subst d-IN-Sup*[*symmetric*])(*simp-all add*: *SUP-apply*[*abs-def*])
also have ... = (*SUP* $g{\in}Y$. *d-IN* (λ*e*. *f e* − *g e*) *x*) **unfolding** *image-image* **..**
also have ... = (*SUP* $g{\in}Y$. *d-IN f x* − *d-IN g x*)
by(*rule SUP-cong*[*OF refl*] *d-IN-diff*)+(*auto intro*: *bounded simp add*: *finite2*)
also have ... = *d-IN f x* − *?rhs2* **by**(*subst SUP-const-minus-ennreal*)(*simp-all add*: *Y*)
also have *d-IN f x* − ... = *?rhs2*
by(*subst diff-diff-ennreal*)(*simp-all add*: *finite bounded1*)
finally show *?IN* **. }**
qed

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*
by(*cases e*)(*simp add*: *flow.simps*)

lemma *flowD-KIR*: ⟦ *flow* Δ *f*; *x* $\neq$ *source* Δ; *x* $\neq$ *sink* Δ ⟧ $\Longrightarrow$ *KIR f x*
by(*simp add*: *flow.simps*)

lemma *flowD-capacity-OUT*: *flow* Δ *f* $\Longrightarrow$ *d-OUT f x* $\leq$ *d-OUT* (*capacity* Δ) *x*
by(*rule d-OUT-mono*)(*erule flowD-capacity*)

lemma *flowD-capacity-IN*: *flow* Δ *f* $\Longrightarrow$ *d-IN f x* $\leq$ *d-IN* (*capacity* Δ) *x*
by(*rule d-IN-mono*)(*erule flowD-capacity*)

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* ⇒ '*v cut* ⇒ *bool*
  for Δ **and** *S*
where *cut*: ⟦ *source* Δ ∈ *S*; *sink* Δ ∉ *S* ⟧ ⟹ *cut* Δ *S*

inductive *orthogonal* :: ('*v*, '*more*) *network-scheme* ⇒ '*v flow* ⇒ '*v cut* ⇒ *bool*
  for Δ *f S*
where
  ⟦ ⋀*x y*. ⟦ *edge* Δ *x y*; *x* ∈ *S*; *y* ∉ *S* ⟧ ⟹ *f* (*x*, *y*) = *capacity* Δ (*x*, *y*);
    ⋀*x y*. ⟦ *edge* Δ *x y*; *x* ∉ *S*; *y* ∈ *S* ⟧ ⟹ *f* (*x*, *y*) = *0* ⟧
  ⟹ *orthogonal* Δ *f S*

lemma *orthogonalD-out*:
  ⟦ *orthogonal* Δ *f S*; *edge* Δ *x y*; *x* ∈ *S*; *y* ∉ *S* ⟧ ⟹ *f* (*x*, *y*) = *capacity* Δ (*x*, *y*)
by(*simp add*: *orthogonal.simps*)

lemma *orthogonalD-in*:
  ⟦ *orthogonal* Δ *f S*; *edge* Δ *x y*; *x* ∉ *S*; *y* ∈ *S* ⟧ ⟹ *f* (*x*, *y*) = *0*
by(*simp add*: *orthogonal.simps*)

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* Δ ≠ *sink* Δ
  and *capacity-outside*: *e* ∉ **E** ⟹ *capacity* Δ *e* = *0*
  and *capacity-finite* [*simp*]: *capacity* Δ *e* ≠ ⊤
begin

lemma *sink-neq-source* [*simp*]: *sink* Δ ≠ *source* Δ
using *source-neq-sink*[*symmetric*] **.**

lemma *countable-V* [*simp*]: *countable* **V**
unfolding **V**-*def* **using** *countable-E* **by** *auto*

lemma *flowD-outside*:
  assumes *g*: *flow* Δ *g*
  shows *e* ∉ **E** ⟹ *g e* = *0*
using *flowD-capacity*[*OF g*, *of e*] *capacity-outside*[*of e*] **by** *simp*

lemma *flowD-finite*:

assumes *flow* Δ *g*
shows $g\ e \neq \top$
using *flowD-capacity*[*OF assms*, *of e*] **by** (*auto simp*: *top-unique*)

lemma *zero-flow* [*simp*]: *flow* Δ (λ-. *0*)
by(*rule flow.intros*) *simp-all*

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* ⇒ '*v vertex* ⇒ *bool*
where
OUT: *edge* Δ *x y* ⟹ *edg* (*Vertex x*) (*Edge x y*)
| *IN*: *edge* Δ *x y* ⟹ *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* ⇒ '*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* ≠ *Edge x z*) ⟹ *split f* (*Vertex x*, *y*) = *0*
by(*cases y*) *auto*

lemma *split-Vertex2-eq-0I*: ($\bigwedge$*z*. *y* ≠ *Edge z x*) ⟹ *split f* (*y*, *Vertex x*) = *0*
by(*cases y*) *simp-all*

lemma *split-Edge1-eq-0I*: ($\bigwedge$*z*. *y* ≠ *Vertex x*) ⟹ *split f* (*Edge z x*, *y*) = *0*
by(*cases y*) *simp-all*

lemma *split-Edge2-eq-0I*: ($\bigwedge$*z*. *y* ≠ *Vertex x*) ⟹ *split f* (*y*, *Edge x z*) = *0*
by(*cases y*) *simp-all*

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)
by(*simp-all add*: Δ''*-def*)

lemma E-Δ'': $\mathbf{E}_{\Delta''}$ = (λ(*x*, *y*). (*Vertex x*, *Edge x y*)) ‘ **E** ∪ (λ(*x*, *y*). (*Edge x y*, *Vertex y*)) ‘ **E**
by(*auto elim*: *edg.cases*)

lemma V-Δ'': $\mathbf{V}_{\Delta''}$ = *Vertex* ‘ **V** ∪ *case-prod Edge* ‘ **E**
by(*auto 4 4 simp add*: *vertex-def elim*!: *edg.cases*)

lemma *inj-on-Edge1* [*simp*]: *inj-on* (λ*x*. *Edge x y*) *A*
by(*simp add*: *inj-on-def*)

lemma *inj-on-Edge2* [*simp*]: *inj-on* (*Edge x*) *A*
by(*simp add*: *inj-on-def*)

lemma *d-IN-split-Vertex* [*simp*]: *d-IN* (*split f*) (*Vertex x*) = *d-IN f x* (**is** *?lhs* = *?rhs*)
proof(*rule trans*)
show *?lhs* = ($\sum^+$ *v'*∈*range* (λ*y*. *Edge y x*). *split f* (*v'*, *Vertex x*))
by(*auto intro*!: *nn-integral-cong split-Vertex2-eq-0I simp add*: *d-IN-def nn-integral-count-space-indicator split*: *split-indicator*)
show ... = *?rhs* **by**(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
qed

lemma *d-OUT-split-Vertex* [*simp*]: *d-OUT* (*split f*) (*Vertex x*) = *d-OUT f x* (**is** *?lhs* = *?rhs*)
proof(*rule trans*)
show *?lhs* = ($\sum^+$ *v'*∈*range* (*Edge x*). *split f* (*Vertex x*, *v'*))
by(*auto intro*!: *nn-integral-cong split-Vertex1-eq-0I simp add*: *d-OUT-def nn-integral-count-space-indicator split*: *split-indicator*)
show ... = *?rhs* **by**(*simp add*: *nn-integral-count-space-reindex d-OUT-def*)
qed

lemma *d-IN-split-Edge* [*simp*]: *d-IN* (*split f*) (*Edge x y*) = *max 0* (*f* (*x*, *y*)) (**is** *?lhs* = *?rhs*)
proof(*rule trans*)
show *?lhs* = ($\sum^+$ *v'*. *split f* (*v'*, *Edge x y*) ∗ *indicator* {*Vertex x*} *v'*)
unfolding *d-IN-def* **by**(*rule nn-integral-cong*)(*simp add*: *split-Edge2-eq-0I split*: *split-indicator*)
show ... = *?rhs* **by**(*simp add*: *max-def*)

qed

lemma *d-OUT-split-Edge* [*simp*]: *d-OUT* (*split f*) (*Edge x y*) = *max 0* (*f* (*x*, *y*))
(**is** *?lhs* = *?rhs*)
proof(*rule trans*)
  show *?lhs* = ($\sum^+$ *v'*. *split f* (*Edge x y*, *v'*) ∗ *indicator* {*Vertex y*} *v'*)
    unfolding *d-OUT-def* **by**(*rule nn-integral-cong*)(*simp add*: *split-Edge1-eq-0I*
split: *split-indicator*)
  show … = *?rhs* **by**(*simp add*: *max-def*)
qed

lemma Δ''-*countable-network*: *countable-network* Δ''
proof
  show *countable* $\mathbf{E}_{\Delta''}$ **unfolding** **E**-Δ'' **by**(*simp*)
  show *source* $\Delta'' \neq$ *sink* Δ'' **by** *auto*
  show *capacity* Δ'' *e* = *0* **if** $e \notin \mathbf{E}_{\Delta''}$ **for** *e* **using** *that*
   by(*cases* (*capacity* Δ, *e*) *rule*: *split.cases*)(*auto simp add*: *capacity-outside*)
  show *capacity* Δ'' *e* ≠ *top* **for** *e* **by**(*cases* (*capacity* Δ, *e*) *rule*: *split.cases*)(*auto*)
qed

interpretation Δ'': *countable-network* Δ'' **by**(*rule* Δ''-*countable-network*)

lemma *flow-split* [*simp*]:
  assumes *flow* Δ *f*
  shows *flow* Δ'' (*split f*)
proof
  show *split f e* ≤ *capacity* Δ'' *e* **for** *e*
    by(*cases* (*f*, *e*) *rule*: *split.cases*)(*auto intro*: *flowD-capacity*[*OF assms*] *intro*:
SUP-upper2 assms)
  show *KIR* (*split f*) *x* **if** *x* ≠ *source* Δ'' *x* ≠ *sink* Δ'' **for** *x*
   using *that* **by**(*cases x*)(*auto dest*: *flowD-KIR*[*OF assms*])
qed

abbreviation (*input*) *collect* :: *'v vertex flow* ⇒ *'v flow*
where *collect f* ≡ (λ(*x*, *y*). *f* (*Edge x y*, *Vertex y*))

lemma *d-OUT-collect*:
  assumes *f*: *flow* Δ'' *f*
  shows *d-OUT* (*collect f*) *x* = *d-OUT f* (*Vertex x*)
proof −
  have *d-OUT* (*collect f*) *x* = ($\sum^+$ *y*. *f* (*Edge x y*, *Vertex y*))
   by(*simp add*: *nn-integral-count-space-reindex d-OUT-def*)
  also have … = ($\sum^+$ *y*∈*range* (*Edge x*). *f* (*Vertex x*, *y*))
  proof(*clarsimp simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-cong*)
   fix *y*
   have ($\sum^+$ *z*. *f* (*Edge x y*, *z*) ∗ *indicator* {*Vertex y*} *z*) = *d-OUT f* (*Edge x y*)
     unfolding *d-OUT-def* **by**(*rule nn-integral-cong*)(*simp split*: *split-indicator*
add: Δ''.*flowD-outside*[*OF f*])
   also have … = *d-IN f* (*Edge x y*) **using** *f* **by**(*rule flowD-KIR*) *simp-all*

also have ... $= (\sum^+ z.\ f\ (z,\ Edge\ x\ y) * indicator\ \{Vertex\ x\}\ z)$
unfolding *d-IN-def* **by**(*rule nn-integral-cong*)(*simp split*: *split-indicator add*: Δ''.*flowD-outside*[*OF f*])
finally show $f\ (Edge\ x\ y,\ Vertex\ y) = f\ (Vertex\ x,\ Edge\ x\ y)$
by(*simp add*: *max-def*)
qed
also have ... = *d-OUT f* (*Vertex x*)
by(*auto intro*!: *nn-integral-cong* Δ''.*flowD-outside*[*OF f*] *simp add*: *nn-integral-count-space-indicator d-OUT-def split*: *split-indicator*)
finally show *?thesis* **.**
qed

lemma *flow-collect* [*simp*]:
assumes *f*: *flow* Δ'' *f*
shows *flow* Δ (*collect f*)
proof
show *collect f e* $\leq$ *capacity* Δ *e* **for** *e* **using** *flowD-capacity*[*OF f*, *of* (*case-prod Edge e*, *Vertex* (*snd e*))]
by(*cases e*)(*simp*)

fix *x*
assume *x*: $x \neq source\ \Delta\ x \neq sink\ \Delta$
have *d-OUT* (*collect f*) *x* = *d-OUT f* (*Vertex x*) **using** *f* **by**(*rule d-OUT-collect*)
also have ... = *d-IN f* (*Vertex x*) **using** *x flowD-KIR*[*OF f*, *of Vertex x*] **by**(*simp*)
also have ... $= (\sum^+ y{\in}range\ (\lambda z.\ Edge\ z\ x).\ f\ (y,\ Vertex\ x))$
by(*auto intro*!: *nn-integral-cong* Δ''.*flowD-outside*[*OF f*] *simp add*: *nn-integral-count-space-indicator d-IN-def split*: *split-indicator*)
also have ... = *d-IN* (*collect f*) *x* **by**(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
finally show *KIR* (*collect f*) *x* **.**
qed

lemma *value-collect*: *flow* Δ'' *f* $\Longrightarrow$ *value-flow* Δ (*collect f*) = *value-flow* Δ'' *f*
by(*simp add*: *d-OUT-collect*)

end

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

6 Webs and currents

record *'v web* = *'v graph* +
weight :: *'v* $\Rightarrow$ *ennreal*
A :: *'v set*
B :: *'v set*

lemma *vertex-weight-update* [*simp*]: *vertex* (*weight-update* *f* Γ) = *vertex* Γ
by(*simp add*: *vertex-def fun-eq-iff*)

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

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

lemma *currentD-weight-OUT*: $current\ \Gamma\ f \Longrightarrow d\text{-}OUT\ f\ x \leq weight\ \Gamma\ x$
by(*simp add*: *current.simps*)

lemma *currentD-weight-IN*: $current\ \Gamma\ f \Longrightarrow d\text{-}IN\ f\ x \leq weight\ \Gamma\ x$
by(*simp add*: *current.simps*)

lemma *currentD-OUT-IN*: $[\![current\ \Gamma\ f;\ x \notin A\ \Gamma\ ]\!] \Longrightarrow d\text{-}OUT\ f\ x \leq d\text{-}IN\ f\ x$
by(*simp add*: *current.simps*)

lemma *currentD-IN*: $[\![current\ \Gamma\ f;\ a \in A\ \Gamma\ ]\!] \Longrightarrow d\text{-}IN\ f\ a = 0$
by(*simp add*: *current.simps*)

lemma *currentD-OUT*: $[\![current\ \Gamma\ f;\ b \in B\ \Gamma\ ]\!] \Longrightarrow d\text{-}OUT\ f\ b = 0$
by(*simp add*: *current.simps*)

lemma *currentD-outside*: $[\![current\ \Gamma\ f;\ \neg\ edge\ \Gamma\ x\ y\ ]\!] \Longrightarrow f\ (x,\ y) = 0$
by(*blast elim*: *current.cases*)

lemma *currentD-outside'*: $[\![current\ \Gamma\ f;\ e \notin \mathbf{E}_\Gamma\ ]\!] \Longrightarrow f\ e = 0$
by(*blast elim*: *current.cases*)

lemma *currentD-OUT-eq-0*:
assumes *current* Γ *f*
shows $d\text{-}OUT\ f\ x = 0 \longleftrightarrow (\forall\, y.\ f\ (x,\ y) = 0)$
by(*simp add*: *d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0*)

lemma *currentD-IN-eq-0*:
assumes *current* Γ *f*
shows $d\text{-}IN\ f\ x = 0 \longleftrightarrow (\forall\, y.\ f\ (y,\ x) = 0)$
by(*simp add*: *d-IN-def nn-integral-0-iff emeasure-count-space-eq-0*)

lemma *current-support-flow*:
fixes Γ (**structure**)
assumes *current* Γ *f*
shows *support-flow f* $\subseteq$ **E**
using *currentD-outside*[*OF assms*] **by**(*auto simp add*: *support-flow.simps intro*: *ccontr*)

lemma *currentD-outside-IN*: $[\![$ *current* Γ *f*; $x \notin \mathbf{V}_\Gamma$ $]\!] \Longrightarrow$ *d-IN f x* = *0*
by(*auto simp add*: *d-IN-def vertex-def nn-integral-0-iff AE-count-space emeasure-count-space-eq-0 dest*: *currentD-outside*)

lemma *currentD-outside-OUT*: $[\![$ *current* Γ *f*; $x \notin \mathbf{V}_\Gamma$ $]\!] \Longrightarrow$ *d-OUT f x* = *0*
by(*auto simp add*: *d-OUT-def vertex-def nn-integral-0-iff AE-count-space emeasure-count-space-eq-0 dest*: *currentD-outside*)

lemma *currentD-weight-in*: *current* Γ *h* $\Longrightarrow$ *h* (*x*, *y*) $\leq$ *weight* Γ *y*
by (*metis order-trans d-IN-ge-point currentD-weight-IN*)

lemma *currentD-weight-out*: *current* Γ *h* $\Longrightarrow$ *h* (*x*, *y*) $\leq$ *weight* Γ *x*
by (*metis order-trans d-OUT-ge-point currentD-weight-OUT*)

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*
proof
show *d-OUT g x* $\leq$ *weight* Γ *x* **for** *x*
using *d-OUT-mono*[*of g x f*, *OF le*] *currentD-weight-OUT*[*OF f*] **by**(*rule order-trans*)
show *d-IN g x* $\leq$ *weight* Γ *x* **for** *x*
using *d-IN-mono*[*of g x f*, *OF le*] *currentD-weight-IN*[*OF f*] **by**(*rule order-trans*)
show *d-IN g a* = *0* **if** $a \in A\ \Gamma$ **for** *a*
using *d-IN-mono*[*of g a f*, *OF le*] *currentD-IN*[*OF f that*] **by** *auto*
show *d-OUT g b* = *0* **if** $b \in B\ \Gamma$ **for** *b*
using *d-OUT-mono*[*of g b f*, *OF le*] *currentD-OUT*[*OF f that*] **by** *auto*
show *g e* = *0* **if** $e \notin$ **E** **for** *e*
using *currentD-outside'*[*OF f that*] *le*[*of e*] **by** *simp*
qed(*blast intro*: *OUT-IN*)+

lemma *current-weight-mono*:
$[\![$ *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*
by(*auto 4 3 elim*!: *current.cases intro*!: *current.intros intro*: *order-trans*)

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

lemma *SINK-0* [*simp*]: *SINK zero-current* = *UNIV*
by(*auto simp add*: *SINK.simps*)

lemma *current-0* [*simp*]: *current* Γ *zero-current*
by(*auto simp add*: *current.simps*)

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

lemma *web-flowD-current*: *web-flow* Γ $f \Longrightarrow$ *current* Γ f
by(*erule web-flow.cases*)

lemma *web-flowD-KIR*: $[\![$ *web-flow* Γ f; $x \notin A\ \Gamma$; $x \notin B\ \Gamma$ $]\!] \Longrightarrow$ *KIR* f x
apply(*cases* $x \in \mathbf{V}_\Gamma$)
 apply(*fastforce elim*!: *web-flow.cases*)
apply(*auto simp add*: *vertex-def d-OUT-def d-IN-def elim*!: *web-flow.cases*)
apply(*subst* (*1 2*) *currentD-outside*[*of* - f]; *auto*)
done

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* Γ f
| *IN*: *d-IN* f $x \geq$ *weight* Γ $x \Longrightarrow x \in$ *SAT* Γ 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* Γ $x \leq 0\}$
by(*auto simp add*: *SAT.simps*)

lemma *SAT-mono*:
  assumes $\bigwedge e.\ f\ e \leq g\ e$
  shows *SAT* Γ $f \subseteq$ *SAT* Γ g
proof
  fix x
  assume $x \in$ *SAT* Γ f
  thus $x \in$ *SAT* Γ g
  proof *cases*
    case *IN*
    also have *d-IN* f $x \leq$ *d-IN* g x **using** *assms* **by**(*rule d-IN-mono*)
    finally show *?thesis* **..**
  qed(*rule SAT.A*)
qed

lemma *SAT-Sup-upper*: $f \in Y \Longrightarrow SAT\ \Gamma\ f \subseteq SAT\ \Gamma\ (Sup\ Y)$
by(*rule SAT-mono*)(*rule Sup-upper*[*THEN le-funD*])

lemma *currentD-SAT*:
  assumes *current* Γ *f*
  shows $x \in SAT\ \Gamma\ f \longleftrightarrow x \in A\ \Gamma \lor d\text{-}IN\ f\ x = weight\ \Gamma\ x$
using *currentD-weight-IN*[*OF assms*, *of x*] **by**(*auto simp add*: *SAT.simps*)

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

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

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?*]:
  $(\bigwedge p.\ path\ \Delta\ (source\ \Delta)\ p\ (sink\ \Delta) \Longrightarrow (\exists z \in set\ p.\ z \in S) \lor source\ \Delta \in S)$
  $\Longrightarrow separating\text{-}network\ \Delta\ S$
by(*auto intro*: *separating*)

lemma *separatingD*:
  $\bigwedge A\ B.\ [\![\ separating\text{-}gen\ G\ A\ B\ S;\ path\ G\ x\ p\ y;\ x \in A;\ y \in B\ ]\!] \Longrightarrow (\exists z \in set\ p.\ z \in S) \lor x \in S$
by(*blast elim*: *separating-gen.cases*)

lemma *separating-left* [*simp*]: $\bigwedge A\ B.\ A \subseteq A' \Longrightarrow separating\text{-}gen\ \Gamma\ A\ B\ A'$
by(*auto simp add*: *separating-gen.simps*)

lemma *separating-weakening*:
  $\bigwedge A\ B.\ [\![\ separating\text{-}gen\ G\ A\ B\ S;\ S \subseteq S'\ ]\!] \Longrightarrow separating\text{-}gen\ G\ A\ B\ S'$
by(*rule separating*; *drule* (*3*) *separatingD*; *blast*)

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

abbreviation *essential-web* :: ($'v$, $'more$) *web-scheme* $\Rightarrow$ $'v$ *set* $\Rightarrow$ $'v$ *set* (‹$\mathcal{E}$1›)

where *essential-web* Γ $S \equiv \{x{\in}S.\ essential\ \Gamma\ (B\ \Gamma)\ S\ x\}$

lemma *essential-weight-update* [*simp*]:
essential (*weight-update f G*) = *essential G*
by(*simp add*: *essential-def fun-eq-iff*)

lemma *not-essentialD*:
$\bigwedge B.\ [\![\ \neg\ essential\ G\ B\ S\ x;\ path\ G\ x\ p\ y;\ y \in B\ ]\!] \Longrightarrow x \neq y \wedge (\exists z{\in}set\ p.\ z \neq x \wedge z \in S)$
by(*simp add*: *essential-def*)

lemma *essentialE* [*elim?*, *consumes 1*, *case-names essential*, *cases pred*: *essential*]:
$\bigwedge B.\ [\![\ essential\ G\ B\ S\ x;\ \bigwedge p\ y.\ [\![\ path\ G\ x\ p\ y;\ y \in B;\ \bigwedge z.\ [\![\ x \neq y;\ z \in set\ p\ ]\!] \Longrightarrow z = x \vee z \notin S\ ]\!] \Longrightarrow thesis\ ]\!] \Longrightarrow thesis$
by(*auto simp add*: *essential-def*)

lemma *essentialI* [*intro?*]:
$\bigwedge B.\ [\![\ path\ G\ x\ p\ y;\ y \in B;\ \bigwedge z.\ [\![\ x \neq y;\ z \in set\ p\ ]\!] \Longrightarrow z = x \vee z \notin S\ ]\!] \Longrightarrow essential\ G\ B\ S\ x$
by(*auto simp add*: *essential-def*)

lemma *essential-vertex*: $\bigwedge B.\ [\![\ essential\ G\ B\ S\ x;\ x \notin B\ ]\!] \Longrightarrow vertex\ G\ x$
by(*auto elim*!: *essentialE simp add*: *vertex-def elim*: *rtrancl-path.cases*)

lemma *essential-BI*: $\bigwedge B.\ x \in B \Longrightarrow essential\ G\ B\ S\ x$
by(*auto simp add*: *essential-def intro*: *rtrancl-path.base*)

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** $path\ \Gamma\ x\ p\ y\ y \in B\ \Gamma\ \bigwedge z.\ [\![\ x \neq y;\ z \in set\ p\ ]\!] \Longrightarrow z = x \vee z \notin S$
using *assms* **by**(*auto elim*: *essentialE*)

lemma *essential-mono*: $\bigwedge B.\ [\![\ essential\ G\ B\ S\ x;\ S' \subseteq S\ ]\!] \Longrightarrow essential\ G\ B\ S'\ x$
by(*auto simp add*: *essential-def*)

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
fix *x y p*
assume *x*: $x \in A$ **and** *y*: $y \in B$ **and** *p*: *path G x p y*
from *separatingD*[*OF assms p x y*] **have** $\exists z \in set\ (x\ \#\ p).\ z \in S$ **by** *auto*
from *split-list-last-prop*[*OF this*] **obtain** *ys z zs* **where** *decomp*: $x\ \#\ p = ys\ @\ z\ \#\ zs$
and *z*: $z \in S$ **and** *last*: $\bigwedge z.\ z \in set\ zs \Longrightarrow z \notin S$ **by** *auto*

from *decomp* **consider** (*empty*) $ys = []$ $x = z$ $p = zs$
| (*Cons*) ys' **where** $ys = x \# ys'$ $p = ys' @ z \# zs$
by(*auto simp add*: *Cons-eq-append-conv*)
then show $(\exists z \in set\ p.\ z \in ?E) \lor x \in ?E$
proof(*cases*)
case *empty*
hence $x \in ?E$ **using** *z p last y* **by**(*auto simp add*: *essential-def*)
thus *?thesis* **..**
next
case (*Cons ys'*)
from *p* **have** *path G z zs y* **unfolding** *Cons* **by**(*rule rtrancl-path-appendE*)
hence $z \in ?E$ **using** *z y last* **by**(*auto simp add*: *essential-def*)
thus *?thesis* **using** *Cons* **by** *auto*
qed
qed

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

abbreviation *roofed* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *'v set* $\Rightarrow$ *'v set* (‹RF1›)
where *roofed* $\Gamma \equiv$ *roofed-gen* Γ $(B\ \Gamma)$

abbreviation *roofed-network* :: $('v, 'more)$ *network-scheme* $\Rightarrow$ *'v set* $\Rightarrow$ *'v set* (‹RF^N1›)
where *roofed-network* $\Delta \equiv$ *roofed-gen* Δ $\{sink\ \Delta\}$

lemma *roofedI* [*intro?*]:
$\bigwedge B.$ $(\bigwedge p\ y.\ [\![$ *path G x p y*; $y \in B$ $]\!] \Longrightarrow (\exists z \in set\ p.\ z \in S) \lor x \in S) \Longrightarrow x \in$ *roofed-gen G B S*
by(*auto simp add*: *roofed-def*)

lemma *not-roofedE*: **fixes** *B*
assumes $x \notin$ *roofed-gen G B S*
obtains *p y* **where** *path G x p y* $y \in B$ $\bigwedge z.\ z \in set\ (x \# p) \Longrightarrow z \notin S$
using *assms* **by**(*auto simp add*: *roofed-def*)

lemma *roofed-greater*: $\bigwedge B.$ $S \subseteq$ *roofed-gen G B S*
by(*auto simp add*: *roofed-def*)

lemma *roofed-greaterI*: $\bigwedge B.$ $x \in S \Longrightarrow x \in$ *roofed-gen G B S*
using *roofed-greater*[*of S G*] **by** *blast*

lemma *roofed-mono*: $\bigwedge B.$ $S \subseteq S' \Longrightarrow$ *roofed-gen G B S* $\subseteq$ *roofed-gen G B S'*
by(*fastforce simp add*: *roofed-def*)

lemma *in-roofed-mono*: $\bigwedge B.$ $[\![\ x \in$ *roofed-gen G B S*; $S \subseteq S'\ ]\!] \Longrightarrow x \in$ *roofed-gen G B S'*
using *roofed-mono*[*THEN subsetD*] **.**

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. $z \in S$) $\vee$ $x \in S$
unfolding *roofed-def* **by** *blast*

lemma *separating-RF-A*:
  fixes *A B*
  assumes *separating-gen G A B X*
  shows $A \subseteq$ *roofed-gen G B X*
by(*rule subsetI roofedI*)+(*erule separatingD*[*OF assms*])

lemma *roofed-idem*: **fixes** *B* **shows** *roofed-gen G B* (*roofed-gen G B S*) = *roofed-gen G B S*
proof(*rule equalityI subsetI roofedI*)+
  fix *x p y*
  assume *x*: $x \in$ *roofed-gen G B* (*roofed-gen G B S*) **and** *p*: *path G x p y* **and** *y*: $y \in B$
  from *roofedD*[*OF x p y*] **obtain** *z* **where** $*$: $z \in$ *set* (x # p) **and** *z*: $z \in$ *roofed-gen G B S* **by** *auto*
  from *split-list*[*OF* $*$] **obtain** *ys zs* **where** *split*: x # p = *ys* @ z # *zs* **by** *blast*
   with *p* **have** p': *path G z zs y* **by**(*auto simp add*: *Cons-eq-append-conv elim*: *rtrancl-path-appendE*)
  from *roofedD*[*OF z p' y*] *split* **show** ($\exists$ $z\in$*set* p. $z \in S$) $\vee$ $x \in S$
    by(*auto simp add*: *Cons-eq-append-conv*)
qed(*rule roofed-mono roofed-greater*)+

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'*
by(*subst roofed-idem*[*symmetric*])(*erule in-roofed-mono*)

lemma *roofed-mono'*: $\bigwedge B$. $S \subseteq$ *roofed-gen G B S'* $\Longrightarrow$ *roofed-gen G B S* $\subseteq$ *roofed-gen G B S'*
by(*rule subsetI*)(*rule in-roofed-mono'*)

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 −
  have $S \subseteq T \cup$ *roofed-gen G B S*
    by (*metis* (*no-types*) *UnCI roofed-greater subsetCE subsetI*)
  then have $S \cup T \subseteq T \cup$ *roofed-gen G B S* $\wedge$ $T \cup$ *roofed-gen G B S* $\subseteq$ *roofed-gen G B* ($S \cup T$)
   by (*metis* (*no-types*) *Un-subset-iff Un-upper2 roofed-greater roofed-mono sup.commute*)
  then show *?thesis*
    by (*metis* (*no-types*) *roofed-idem roofed-mono subset-antisym sup.commute*)
qed

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(rule equalityI)
  show ?rhs ⊆ ?lhs by(rule roofed-mono)(blast intro: roofed-greaterI)
  show ?lhs ⊆ ?rhs by(rule roofed-mono')(blast intro: in-roofed-mono)
qed

lemma RF-essential: fixes Γ (structure) shows RF (ℰ S) = RF S
proof(intro set-eqI iffI)
  fix x
  assume RF: x ∈ RF S
  show x ∈ RF (ℰ S)
  proof
    fix p y
    assume p: path Γ x p y and y: y ∈ B Γ
    from roofedD[OF RF this] have ∃z∈set (x # p). z ∈ S by auto
    from split-list-last-prop[OF this] obtain ys z zs where decomp: x # p = ys @
z # zs
      and z: z ∈ S and last: ⋀z. z ∈ set zs ⟹ z ∉ S by auto
    from decomp consider (empty) ys = [] x = z p = zs
      | (Cons) ys' where ys = x # ys' p = ys' @ z # zs
      by(auto simp add: Cons-eq-append-conv)
    then show (∃z∈set p. z ∈ ℰ S) ∨ x ∈ ℰ S
    proof(cases)
      case empty
      hence x ∈ ℰ S using z p last y by(auto simp add: essential-def)
      thus ?thesis ..
    next
      case (Cons ys')
      from p have path Γ z zs y unfolding Cons by(rule rtrancl-path-appendE)
      hence z ∈ ℰ S using z y last by(auto simp add: essential-def)
      thus ?thesis using Cons by auto
    qed
  qed
qed(blast intro: in-roofed-mono)

lemma essentialE-RF:
  fixes Γ (structure) and B
  assumes essential Γ B S x
  obtains p y where path Γ x p y y ∈ B distinct (x # p) ⋀z. z ∈ set p ⟹ z ∉
roofed-gen Γ B S
proof -
  from assms obtain p y where p: path Γ x p y and y: y ∈ B
    and bypass: ⋀z. ⟦ x ≠ y; z ∈ set p ⟧ ⟹ z = x ∨ z ∉ S by(rule essentialE)
blast
  from p obtain p' where p': path Γ x p' y and distinct: distinct (x # p')
    and subset: set p' ⊆ set p by(rule rtrancl-path-distinct)
  { fix z
    assume z: z ∈ set p'
    hence y ∈ set p' using rtrancl-path-last[OF p', symmetric] p'
      by(auto elim: rtrancl-path.cases intro: last-in-set)
```

with *distinct z subset* **have** *neq*: $x \neq y$ **and** $z \in$ *set p* **by**(*auto*)
from *bypass*[*OF this*] *z distinct* **have** $z \notin S$ **by** *auto*

have $z \notin$ *roofed-gen* Γ B S
proof
assume z': $z \in$ *roofed-gen* Γ B S
from *split-list*[*OF z*] **obtain** *ys zs* **where** *decomp*: $p' = ys$ @ z # zs **by** *blast*
with p' **have** *path* Γ *z zs y* **by**(*auto elim*: *rtrancl-path-appendE*)
from *roofedD*[*OF z′ this y*] ‹$z \notin S$› **obtain** z' **where** $z' \in$ *set zs* $z' \in S$ **by** *auto*
with *bypass*[*of z′*] *neq decomp subset distinct* **show** *False* **by** *auto*
qed }
with p' *y distinct* **show** *thesis* **..**
qed

lemma $\mathcal{E}$*-E-RF*:
fixes Γ (**structure**)
assumes $x \in \mathcal{E}$ S
obtains *p y* **where** *path* Γ *x p y* $y \in B$ Γ *distinct* (x # p) $\bigwedge z.\ z \in$ *set p* $\Longrightarrow z \notin RF\ S$
using *assms* **by**(*auto elim*: *essentialE-RF*)

lemma *in-roofed-essentialD*:
fixes Γ (**structure**)
assumes *RF*: $x \in RF\ S$
and *ess*: *essential* Γ (B Γ) S x
shows $x \in S$
proof −
from *ess* **obtain** *p y* **where** *p*: *path* Γ *x p y* **and** *y*: $y \in B$ Γ **and** *distinct* (x # p)
and *bypass*: $\bigwedge z.\ z \in$ *set p* $\Longrightarrow z \notin S$ **by**(*rule essentialE-RF*)(*auto intro*: *roofed-greaterI*)
from *roofedD*[*OF RF p y*] *bypass* **show** $x \in S$ **by** *auto*
qed

lemma *separating-RF*: **fixes** Γ (**structure**) **shows** *separating* Γ (*RF S*) $\longleftrightarrow$ *separating* Γ S
proof
assume *sep*: *separating* Γ (*RF S*)
show *separating* Γ S
proof
fix *x y p*
assume *p*: *path* Γ *x p y* **and** *x*: $x \in A$ Γ **and** *y*: $y \in B$ Γ
from *separatingD*[*OF sep p x y*] **have** $\exists z \in$ *set* (x # p). $z \in RF\ S$ **by** *auto*
from *split-list-last-prop*[*OF this*] **obtain** *ys z zs* **where** *split*: x # p = *ys* @ z # *zs*
and *z*: $z \in RF\ S$ **and** *bypass*: $\bigwedge z'.\ z' \in$ *set zs* $\Longrightarrow z' \notin RF\ S$ **by** *auto*
from *p split* **have** *path* Γ *z zs y* **by**(*cases ys*)(*auto elim*: *rtrancl-path-appendE*)
hence *essential* Γ (B Γ) S z **using** *y*

by(*rule essentialI*)(*auto dest*: *bypass intro*: *roofed-greaterI*)
with z **have** $z \in S$ **by**(*rule in-roofed-essentialD*)
with *split* **show** $(\exists\, z{\in}set\ p.\ z \in S) \vee x \in S$ **by**(*cases ys*)*auto*
qed
qed(*blast intro*: *roofed-greaterI separating-weakening*)

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

lemma *roofed-circI*: **fixes** Γ (**structure**) **shows**
$[\![\ x \in RF\ T;\ x \in T \Longrightarrow \neg$ *essential* $\Gamma\ (B\ \Gamma)\ T\ x\ ]\!] \Longrightarrow x \in RF^\circ\ T$
by(*simp add*: *roofed-circ-def*)

lemma *roofed-circE*:
fixes Γ (**structure**)
assumes $x \in RF^\circ\ T$
obtains $x \in RF\ T$ $\neg$ *essential* $\Gamma\ (B\ \Gamma)\ T\ x$
using *assms* **by**(*auto simp add*: *roofed-circ-def intro*: *in-roofed-essentialD*)

lemma $\mathcal{E}$-$\mathcal{E}$: **fixes** Γ (**structure**) **shows** $\mathcal{E}\ (\mathcal{E}\ S) = \mathcal{E}\ S$
by(*auto intro*: *essential-mono*)

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

lemma *essential-RF*: **fixes** B
shows *essential G B* (*roofed-gen G B S*) = *essential G B S* (**is** *essential - - ?RF* = -)
proof(*intro ext iffI*)
show *essential G B S x* **if** *essential G B ?RF x* **for** x **using** *that*
by(*rule essential-mono*)(*blast intro*: *roofed-greaterI*)
show *essential G B ?RF x* **if** *essential G B S x* **for** x
using *that* **by**(*rule essentialE-RF*)(*erule* (*1*) *essentialI*, *blast*)
qed

lemma $\mathcal{E}$-*RF*: **fixes** Γ (**structure**) **shows** $\mathcal{E}\ (RF\ S) = \mathcal{E}\ S$
by(*auto dest*: *in-roofed-essentialD simp add*: *essential-RF intro*: *roofed-greaterI*)

lemma *essential*-$\mathcal{E}$: **fixes** Γ (**structure**) **shows** *essential* $\Gamma\ (B\ \Gamma)\ (\mathcal{E}\ S)$ = *essential* $\Gamma\ (B\ \Gamma)\ S$
by(*subst essential-RF*[*symmetric*])(*simp only*: *RF-essential essential-RF*)

lemma *RF-in-B*: **fixes** Γ (**structure**) **shows** $x \in B\ \Gamma \Longrightarrow x \in RF\ S \longleftrightarrow x \in S$
by(*auto intro*: *roofed-greaterI dest*: *roofedD*[*OF - rtrancl-path.base*])

lemma *RF-circ-edge-forward*:
fixes Γ (**structure**)
assumes x: $x \in RF^\circ\ S$
and *edge*: *edge* Γ x y

shows $y \in RF\ S$
proof
fix $p\ z$
assume p: $path\ \Gamma\ y\ p\ z$ **and** z: $z \in B\ \Gamma$
from x **have** rf: $x \in RF\ S$ **and** $ness$: $x \notin \mathcal{E}\ S$ **by**(*auto elim*: *roofed-circE*)
show $(\exists z{\in}set\ p.\ z \in S) \vee y \in S$
proof(*cases* $\exists z'{\in}set\ (y \ \#\ p).\ z' \in S$)
case *False*
from *edge p* **have** p': $path\ \Gamma\ x\ (y \ \#\ p)\ z$ **..**
from *roofedD*[*OF rf this z*] *False* **have** $x \in S$ **by** *auto*
moreover have $essential\ \Gamma\ (B\ \Gamma)\ S\ x$ **using** p' *False* z **by**(*auto intro*!: *essentialI*)
ultimately have $x \in \mathcal{E}\ S$ **by** *simp*
with *ness* **show** *?thesis* **by** *contradiction*
qed *auto*
qed

6.3 Waves

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

lemma *wave-0* [*simp*]: *wave* Γ *zero-current*
by *rule simp-all*

lemma *waveD-separating*: $wave\ \Gamma\ f \Longrightarrow separating\ \Gamma\ (TER_\Gamma\ f)$
by(*simp add*: *wave.simps*)

lemma *waveD-OUT*: $[\![\ wave\ \Gamma\ f;\ x \notin RF_\Gamma\ (TER_\Gamma\ f)\ ]\!] \Longrightarrow d\text{-}OUT\ f\ x = 0$
by(*simp add*: *wave.simps*)

lemma *wave-A-in-RF*: **fixes** Γ (**structure**)
shows $[\![\ wave\ \Gamma\ f;\ x \in A\ \Gamma\ ]\!] \Longrightarrow x \in RF\ (TER\ f)$
by(*auto intro*!: *roofedI dest*!: *waveD-separating separatingD*)

lemma *wave-not-RF-IN-zero*:
fixes Γ (**structure**)
assumes f: $current\ \Gamma\ f$
and w: $wave\ \Gamma\ f$
and x: $x \notin RF\ (TER\ f)$
shows $d\text{-}IN\ f\ x = 0$
proof −
from x **obtain** $p\ z$ **where** z: $z \in B\ \Gamma$ **and** p: $path\ \Gamma\ x\ p\ z$
and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin TER\ f\ x \notin TER\ f$

by(*clarsimp simp add*: *roofed-def*)
have $f\ (y,\ x) = 0$ **for** y
proof(*cases edge* Γ *y x*)
case *edge*: *True*
have *d-OUT f y* = *0*
proof(*cases y* ∈ *TER f*)
case *False*
with *z p bypass edge* **have** *y* ∉ *RF* (*TER f*)
by(*auto simp add*: *roofed-def intro*: *rtrancl-path.step intro*!: *exI rev-bexI*)
thus *d-OUT f y* = *0* **by**(*rule waveD-OUT*[*OF w*])
qed(*auto simp add*: *SINK.simps*)
moreover have *f* (*y*, *x*) ≤ *d-OUT f y* **by** (*rule d-OUT-ge-point*)
ultimately show *?thesis* **by** *simp*
qed(*simp add*: *currentD-outside*[*OF f*])
then show *d-IN f x* = *0* **unfolding** *d-IN-def*
by(*simp add*: *nn-integral-0-iff emeasure-count-space-eq-0*)
qed

lemma *current-Sup*:
fixes Γ (**structure**)
assumes *chain*: *Complete-Partial-Order.chain* (≤) *Y*
and *Y*: *Y* ≠ {}
and *current*: ⋀*f*. *f* ∈ *Y* ⟹ *current* Γ *f*
and *countable* [*simp*]: *countable* (*support-flow* (*Sup Y*))
shows *current* Γ (*Sup Y*)
proof(*rule*, *goal-cases*)
case (*1 x*)
have *d-OUT* (*Sup Y*) *x* = (*SUP f*∈*Y*. *d-OUT f x*) **using** *chain Y* **by**(*simp add*: *d-OUT-Sup*)
also have ... ≤ *weight* Γ *x* **using** *1*
by(*intro SUP-least*)(*auto dest*!: *current currentD-weight-OUT*)
finally show *?case* **.**
next
case (*2 x*)
have *d-IN* (*Sup Y*) *x* = (*SUP f*∈*Y*. *d-IN f x*) **using** *chain Y* **by**(*simp add*: *d-IN-Sup*)
also have ... ≤ *weight* Γ *x* **using** *2*
by(*intro SUP-least*)(*auto dest*!: *current currentD-weight-IN*)
finally show *?case* **.**
next
case (*3 x*)
have *d-OUT* (*Sup Y*) *x* = (*SUP f*∈*Y*. *d-OUT f x*) **using** *chain Y* **by**(*simp add*: *d-OUT-Sup*)
also have ... ≤ (*SUP f*∈*Y*. *d-IN f x*) **using** *3*
by(*intro SUP-mono*)(*auto dest*: *current currentD-OUT-IN*)
also have ... = *d-IN* (*Sup Y*) *x* **using** *chain Y* **by**(*simp add*: *d-IN-Sup*)
finally show *?case* **.**
next
case (*4 a*)

have *d-IN* (*Sup Y*) *a* = (*SUP* *f*∈*Y*. *d-IN f a*) **using** *chain Y* **by**(*simp add*: *d-IN-Sup*)
also have ... = (*SUP* *f*∈*Y*. *0*) **using** *4* **by**(*intro SUP-cong*)(*auto dest*!: *current currentD-IN*)
also have ... = *0* **using** *Y* **by** *simp*
finally show *?case* **.**
next
case (*5 b*)
have *d-OUT* (*Sup Y*) *b* = (*SUP* *f*∈*Y*. *d-OUT f b*) **using** *chain Y* **by**(*simp add*: *d-OUT-Sup*)
also have ... = (*SUP* *f*∈*Y*. *0*) **using** *5* **by**(*intro SUP-cong*)(*auto dest*!: *current currentD-OUT*)
also have ... = *0* **using** *Y* **by** *simp*
finally show *?case* **.**
next
fix *e*
assume $e \notin$ **E**
from *currentD-outside'*[*OF current this*] **have** *f e* = *0* **if** $f \in Y$ **for** *f* **using** *that* **by** *simp*
hence *Sup Y e* = (*SUP* -∈*Y*. *0*) **by**(*auto intro*: *SUP-cong*)
then show *Sup Y e* = *0* **using** *Y* **by**(*simp*)
qed

lemma *wave-lub*: — Lemma 4.3
fixes Γ (**structure**)
assumes *chain*: *Complete-Partial-Order.chain* (≤) *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
{ fix *x y p*
assume *p*: *path* Γ *x p y* **and** *y*: $y \in B\ \Gamma$
define *P* **where** $P = \{x\} \cup$ *set p*

let *?f* = λ*f*. *SINK f* ∩ *P*
have *Complete-Partial-Order.chain* (⊇) (*?f* ‘ *Y*) **using** *chain*
by(*rule chain-imageI*)(*auto dest*: *SINK-mono'*)
moreover have ... ⊆ *Pow P* **by** *auto*
hence *finite* (*?f* ‘ *Y*) **by**(*rule finite-subset*)(*simp add*: *P-def*)
ultimately have (⋂(*?f* ‘ *Y*)) ∈ *?f* ‘ *Y*
by(*rule ccpo.in-chain-finite*[*OF complete-lattice-ccpo-dual*])(*simp add*: *Y*)
then obtain *f* **where** *f*: $f \in Y$ **and** *eq*: ⋂(*?f* ‘ *Y*) = *?f f* **by** *clarify*
hence ∗: (⋂*f*∈*Y*. *SINK f*) ∩ *P* = *SINK f* ∩ *P* **by**(*clarsimp simp add*: *prod-lub-def Y*)+
{ fix *g*
assume $g \in Y$ $f \le g$
with ∗ **have** (⋂*f*∈*Y*. *SINK f*) ∩ *P* = *SINK g* ∩ *P* **by**(*blast dest*: *SINK-mono'*)
then have *TER* (*Sup Y*) ∩ *P* ⊇ *TER g* ∩ *P*

```
      using SAT-Sup-upper[OF ‹g ∈ Y›, of Γ] SINK-Sup[OF chain Y countable]
by blast }
    with f have ∃f∈Y. ∀g∈Y. g ≥ f ⟶ TER g ∩ P ⊆ TER (Sup Y) ∩ P by
blast }
  note subset = this

  show separating Γ (TER (Sup Y))
  proof
    fix x y p
    assume *: path Γ x p y y ∈ B Γ and x ∈ A Γ
    let ?P = {x} ∪ set p
    from subset[OF *] obtain f where f:f ∈ Y
      and subset: TER f ∩ ?P ⊆ TER (Sup Y) ∩ ?P by blast
    from wave[OF f] have TER f ∩ ?P ≠ {} using * ‹x ∈ A Γ›
      by(auto simp add: wave.simps dest: separatingD)
    with subset show (∃z∈set p. z ∈ TER (Sup Y)) ∨ x ∈ TER (Sup Y) by blast
  qed

  fix x
  assume x ∉ RF (TER (Sup Y))
  then obtain p y where y: y ∈ B Γ
    and p: path Γ x p y
    and ter: TER (Sup Y) ∩ ({x} ∪ set p) = {} by(auto simp add: roofed-def)
  let ?P = {x} ∪ set p
  from subset[OF p y] obtain f where f: f ∈ Y
    and subset: ⋀g. ⟦ g ∈ Y; f ≤ g ⟧ ⟹ TER g ∩ ?P ⊆ TER (Sup Y) ∩ ?P by
blast

  { fix g
    assume g: g ∈ Y
    with chain f have f ≤ g ∨ g ≤ f by(rule chainD)
    hence d-OUT g x = 0
    proof
      assume f ≤ g
      from subset[OF g this] ter have TER g ∩ ?P = {} by blast
      with p y have x ∉ RF (TER g) by(auto simp add: roofed-def)
      with wave[OF g] show ?thesis by(blast elim: wave.cases)
    next
      assume g ≤ f
      from subset ter f have TER f ∩ ?P = {} by blast
      with y p have x ∉ RF (TER f) by(auto simp add: roofed-def)
      with wave[OF f] have d-OUT f x = 0 by(blast elim: wave.cases)
       moreover have d-OUT g x ≤ d-OUT f x using ‹g ≤ f›[THEN le-funD]
by(rule d-OUT-mono)
      ultimately show ?thesis by simp
    qed }
  thus d-OUT (Sup Y) x = 0 using chain Y by(simp add: d-OUT-Sup)
qed
```

```
lemma ex-maximal-wave: — Corollary 4.4
  fixes Γ (structure)
  assumes countable: countable E
  shows ∃f. current Γ f ∧ wave Γ f ∧ (∀w. current Γ w ∧ wave Γ w ∧ f ≤ w
⟶ f = w)
proof −
  define Field-r where Field-r = {f. current Γ f ∧ wave Γ f}
  define r where r = {(f, g). f ∈ Field-r ∧ g ∈ Field-r ∧ f ≤ g}
  have Field-r: Field r = Field-r by(auto simp add: Field-def r-def)

  have Partial-order r unfolding order-on-defs
    by(auto intro!: refl-onI transI antisymI simp add: Field-r r-def Field-def)
  hence ∃m∈Field r. ∀a∈Field r. (m, a) ∈ r ⟶ a = m
  proof(rule Zorns-po-lemma)
    fix Y
    assume Y ∈ Chains r
    hence Y: Complete-Partial-Order.chain (≤) Y
      and w: ⋀f. f ∈ Y ⟹ wave Γ f
      and f: ⋀f. f ∈ Y ⟹ current Γ f
      by(auto simp add: Chains-def r-def chain-def Field-r-def)
    show ∃w ∈ Field r. ∀f ∈ Y. (f, w) ∈ r
    proof(cases Y = {})
      case True
      have zero-current ∈ Field r by(simp add: Field-r Field-r-def)
      with True show ?thesis by blast
    next
      case False
      have support-flow (Sup Y) ⊆ E by(auto simp add: support-flow-Sup elim!:
support-flow.cases dest!: f dest: currentD-outside)
      hence c: countable (support-flow (Sup Y)) using countable by(rule count-
able-subset)
      with Y False f w have Sup Y ∈ Field r unfolding Field-r Field-r-def
        by(blast intro: wave-lub current-Sup)
      moreover then have (f, Sup Y) ∈ r if f ∈ Y for f using w[OF that] f[OF
that] that unfolding Field-r
        by(auto simp add: r-def Field-r-def intro: Sup-upper)
      ultimately show ?thesis by blast
    qed
  qed
  thus ?thesis by(simp add: Field-r Field-r-def)(auto simp add: r-def Field-r-def)
qed

lemma essential-leI:
  fixes Γ (structure)
  assumes g: current Γ g and w: wave Γ g
  and le: ⋀e. f e ≤ g e
  and x: x ∈ ℰ (TER g)
  shows essential Γ (B Γ) (TER f) x
proof −
```

from x **obtain** p y **where** p: *path* Γ x p y **and** y: $y \in B\ \Gamma$ **and** *distinct*: *distinct* $(x \ \# \ p)$
and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF\ (TER\ g)$ **by**(*rule* $\mathcal{E}$*-E-RF*) *blast*
show *?thesis* **using** p y
proof
fix z
assume $z \in set\ p$
hence z: $z \notin RF\ (TER\ g)$ **by**(*auto dest*: *bypass*)
with w **have** *OUT*: *d-OUT* g $z = 0$ **and** *IN*: *d-IN* g $z = 0$ **by**(*rule waveD-OUT wave-not-RF-IN-zero*[*OF* g])+
with z **have** $z \notin A\ \Gamma$ *weight* Γ $z > 0$ **by**(*auto intro*!: *roofed-greaterI simp add*: *SAT.simps SINK.simps*)
moreover from *IN d-IN-mono*[*of* f z g, *OF le*] **have** *d-IN* f $z \leq 0$ **by**(*simp*)
ultimately have $z \notin TER\ f$ **by**(*auto simp add*: *SAT.simps*)
then show $z = x \lor z \notin TER\ f$ **by** *simp*
qed
qed

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
show *subset*: $\mathcal{E}\ (TER\ g) \subseteq \mathcal{E}\ (TER\ f)$
proof
fix x
assume x: $x \in \mathcal{E}\ (TER\ g)$
hence $x \in TER\ f$ **using** *subset* **by** *blast*
moreover have *essential* Γ $(B\ \Gamma)$ $(TER\ f)$ x **using** g w *le* x **by**(*rule essential-leI*)
ultimately show $x \in \mathcal{E}\ (TER\ f)$ **by** *simp*
qed

show $\ldots \subseteq \mathcal{E}\ (TER\ g)$
proof
fix x
assume x: $x \in \mathcal{E}\ (TER\ f)$
hence $x \in TER\ f$ **by** *auto*
hence $x \in RF\ (TER\ g)$
proof(*rule contrapos-pp*)
assume x: $x \notin RF\ (TER\ g)$
with w **have** *OUT*: *d-OUT* g $x = 0$ **and** *IN*: *d-IN* g $x = 0$ **by**(*rule waveD-OUT wave-not-RF-IN-zero*[*OF* g])+
with x **have** $x \notin A\ \Gamma$ *weight* Γ $x > 0$ **by**(*auto intro*!: *roofed-greaterI simp add*: *SAT.simps SINK.simps*)
moreover from *IN d-IN-mono*[*of* f x g, *OF le*] **have** *d-IN* f $x \leq 0$ **by**(*simp*)
ultimately show $x \notin TER\ f$ **by**(*auto simp add*: *SAT.simps*)

qed
moreover have $x \notin RF^\circ$ (*TER g*)
proof
assume $x \in RF^\circ$ (*TER g*)
hence *RF*: $x \in$ *RF* ($\mathcal{E}$ (*TER g*)) **and** *not-E*: $x \notin \mathcal{E}$ (*TER g*)
unfolding *RF-essential* **by**(*simp-all add*: *roofed-circ-def*)
from *x* **obtain** *p y* **where** *p*: *path* Γ *x p y* **and** *y*: $y \in B\ \Gamma$ **and** *distinct*: *distinct* (*x* # *p*)
and *bypass*: $\bigwedge z.\ z \in$ *set p* $\Longrightarrow z \notin$ *RF* (*TER f*) **by**(*rule* $\mathcal{E}$*-E-RF*) *blast*
from *roofedD*[*OF RF p y*] *not-E* **obtain** *z* **where** $z \in$ *set p* $z \in \mathcal{E}$ (*TER g*) **by** *blast*
with *subset bypass*[*of z*] **show** *False* **by**(*auto intro*: *roofed-greaterI*)
qed
ultimately show $x \in \mathcal{E}$ (*TER g*) **by**(*simp add*: *roofed-circ-def*)
qed
qed

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-OUT f a* $<$ *weight* Γ *a*; $\varepsilon <$ *weight* Γ *a* $-$ *d-OUT f a* $]\!] \Longrightarrow$ *hindrance-by* Γ *f* ε

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-OUT f a* $<$ *weight* Γ *a* $]\!] \Longrightarrow$ *hindrance* Γ *f*

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

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

lemma *hindrance-into-hindrance-by*:
assumes *hindrance* Γ *f*
shows $\exists \varepsilon > 0$. *hindrance-by* Γ *f* ε
using *assms*
proof *cases*
case (*hindrance a*)
let $?\varepsilon$ = *if weight* Γ *a* $= \top$ *then 1 else* (*weight* Γ *a* $-$ *d-OUT f a*) / *2*
from ‹*d-OUT f a* $<$ *weight* Γ *a*› **have** *weight* Γ *a* $-$ *d-OUT f a* > 0 *weight* Γ *a*

$\neq \top \Longrightarrow$ *weight* Γ *a* $-$ *d-OUT f a* $< \top$
  by(*simp-all add*: *diff-gr0-ennreal less-top diff-less-top-ennreal*)
 from *ennreal-mult-strict-left-mono*[*of 1 2*, *OF - this*]
 have *0* $<$ *?ε* **and** *?ε* $<$ *weight* Γ *a* $-$ *d-OUT f a* **using** ‹*d-OUT f a* $<$ *weight* Γ *a*›
  by(*auto intro*!: *diff-gr0-ennreal simp*: *ennreal-zero-less-divide divide-less-ennreal*)
 with *hindrance* **show** *?thesis* **by**(*auto intro*!: *hindrance-by.intros*)
qed

lemma *hindrance-by-into-hindrance*: *hindrance-by* Γ *f* $\varepsilon \Longrightarrow$ *hindrance* Γ *f*
by(*blast elim*: *hindrance-by.cases intro*: *hindrance.intros*)

lemma *hindrance-conv-hindrance-by*: *hindrance* Γ *f* $\longleftrightarrow$ ($\exists \varepsilon > 0$. *hindrance-by* Γ *f* ε)
by(*blast intro*: *hindrance-into-hindrance-by hindrance-by-into-hindrance*)

lemma *hindered-into-hindered-by*: *hindered* $\Gamma \Longrightarrow \exists \varepsilon > 0$. *hindered-by* Γ ε
by(*blast intro*: *hindered-by.intros elim*: *hindered.cases dest*: *hindrance-into-hindrance-by*)

lemma *hindered-by-into-hindered*: *hindered-by* Γ $\varepsilon \Longrightarrow$ *hindered* Γ
by(*blast elim*: *hindered-by.cases intro*: *hindered.intros dest*: *hindrance-by-into-hindrance*)

lemma *hindered-conv-hindered-by*: *hindered* $\Gamma \longleftrightarrow$ ($\exists \varepsilon > 0$. *hindered-by* Γ ε)
by(*blast intro*: *hindered-into-hindered-by hindered-by-into-hindered*)

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

lemma *looseD-hindrance*: *loose* $\Gamma \Longrightarrow \neg$ *hindrance* Γ *zero-current*
by(*simp add*: *loose.simps*)

lemma *looseD-wave*:
 $[\![$ *loose* Γ; *current* Γ *f*; *wave* Γ *f* $]\!] \Longrightarrow f =$ *zero-current*
by(*simp add*: *loose.simps*)

lemma *loose-unhindered*:
 fixes Γ (**structure**)
 assumes *loose* Γ
 shows $\neg$ *hindered* Γ
apply *auto*
 apply(*erule hindered.cases*)
apply(*frule* (*1*) *looseD-wave*[*OF assms*])
apply *simp*
using *looseD-hindrance*[*OF assms*]
by *simp*

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

private lemma *essential-eq*: *essential* Γ = *essential* Γ'
by(*simp add*: *fun-eq-iff essential-def*)

qualified lemma *TER-eq*: $TER_\Gamma\ f = TER_{\Gamma'}\ f$
apply(*auto simp add*: *SINK.simps SAT.simps*)
apply(*erule contrapos-np*; *drule weight-eq*; *simp*)+
done

qualified lemma *separating-eq*: *separating-gen* Γ = *separating-gen* Γ'
by(*intro ext iffI*; *rule separating-gen.intros*; *drule separatingD*; *simp*)

qualified lemma *roofed-eq*: $\bigwedge B.$ *roofed-gen* $\Gamma\ B\ S$ = *roofed-gen* $\Gamma'\ B\ S$
by(*simp add*: *roofed-def*)

lemma *wave-eq-web*: — Observation 4.6
  wave $\Gamma\ f \longleftrightarrow$ *wave* $\Gamma'\ f$
by(*simp add*: *wave.simps separating-eq TER-eq roofed-eq*)

lemma *current-mono-web*: *current* $\Gamma'\ f \Longrightarrow$ *current* $\Gamma\ f$
apply(*rule current*, *simp-all add*: *currentD-OUT-IN currentD-IN currentD-OUT currentD-outside'*)
subgoal for x **by**(*cases* $x \in A\ \Gamma'$)(*auto dest*!: *weight-eq weight-le dest*: *currentD-weight-OUT intro*: *order-trans*)
subgoal for x **by**(*cases* $x \in A\ \Gamma'$)(*auto dest*!: *weight-eq weight-le dest*: *currentD-weight-IN intro*: *order-trans*)
done

lemma *hindrance-mono-web*: *hindrance* $\Gamma'\ f \Longrightarrow$ *hindrance* $\Gamma\ f$
apply(*erule hindrance.cases*)
apply(*rule hindrance*)
  apply *simp*
 apply(*unfold TER-eq*, *simp add*: *essential-eq*)
apply(*auto dest*!: *weight-le*)
done

lemma *hindered-mono-web*: *hindered* $\Gamma' \Longrightarrow$ *hindered* Γ
apply(*erule hindered.cases*)
apply(*rule hindered.intros*)
  apply(*erule hindrance-mono-web*)
 apply(*erule current-mono-web*)
apply(*simp add*: *wave-eq-web*)

done

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*:
$[\![\bigwedge x.\ [\![x \in S;\ x \notin A\ \Gamma]\!] \Longrightarrow weight\ \Gamma\ x \leq d\text{-}IN\ f\ x;$
$\bigwedge x.\ [\![x \in S;\ x \in A\ \Gamma;\ x \notin B\ \Gamma]\!] \Longrightarrow d\text{-}OUT\ f\ x = weight\ \Gamma\ x;$
$\bigwedge u\ v.\ [\![v \in RF\ S;\ u \notin RF^{\circ}\ S]\!] \Longrightarrow f\ (u, v) = 0]\!]$
$\Longrightarrow orthogonal\text{-}current\ \Gamma\ f\ S$

lemma *orthogonal-currentD-SAT*: $[\![orthogonal\text{-}current\ \Gamma\ f\ S;\ x \in S]\!] \Longrightarrow x \in SAT\ \Gamma\ f$
by(*auto elim*!: *orthogonal-current.cases intro*: *SAT.intros*)

lemma *orthogonal-currentD-A*: $[\![orthogonal\text{-}current\ \Gamma\ f\ S;\ x \in S;\ x \in A\ \Gamma;\ x \notin B\ \Gamma]\!] \Longrightarrow d\text{-}OUT\ f\ x = weight\ \Gamma\ x$
by(*auto elim*: *orthogonal-current.cases*)

lemma *orthogonal-currentD-in*: $[\![orthogonal\text{-}current\ \Gamma\ f\ S;\ v \in RF_{\Gamma}\ S;\ u \notin RF^{\circ}{}_{\Gamma}\ S]\!] \Longrightarrow f\ (u, v) = 0$
by(*auto elim*: *orthogonal-current.cases*)

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

lemma *linkageD*: $[\![linkage\ \Gamma\ f;\ x \in A\ \Gamma]\!] \Longrightarrow d\text{-}OUT\ f\ x = weight\ \Gamma\ x$
by(*rule linkage.cases*)

abbreviation *linkable* :: $('v, 'more)$ *web-scheme* $\Rightarrow$ *bool*

where *linkable* $\Gamma \equiv \exists f.$ *web-flow* Γ f $\wedge$ *linkage* Γ 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; $g \leq f$; $\bigwedge x.$ $\llbracket$ $x \in RF^\circ$ (*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 $e \leq f$ 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) $-$ $A\ \Gamma$
using *assms* **by**(*blast elim*: *trimming.cases dest*: *le-funD*)+

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
proof $-$
  define F **where** $F = \{g.$ *current* Γ g $\wedge$ *wave* Γ g $\wedge$ $g \leq f$ $\wedge$ $\mathcal{E}$ (*TER* g) $=$ $\mathcal{E}$ (*TER* f)$\}$
  define *leq* **where** *leq* $=$ *restrict-rel* F $\{(g, g').\ g' \leq g\}$
  have *in-F* [*simp*]: $g \in F \longleftrightarrow$ *current* Γ g $\wedge$ *wave* Γ g $\wedge$ ($\forall e.$ g $e \leq f$ e) $\wedge$ $\mathcal{E}$ (*TER* g) $=$ $\mathcal{E}$ (*TER* f) **for** g
    by(*simp add*: *F-def le-fun-def*)

  have *f-finite* [*simp*]: f $e \neq \top$ **for** e
  proof(*cases* e)
    case (*Pair* x y)
    have f $(x, y) \leq$ *d-IN* f y **by** (*rule d-IN-ge-point*)
    also have $\ldots \leq$ *weight* Γ y **by**(*rule currentD-weight-IN*[*OF* f])
    also have $\ldots < \top$ **by**(*simp add*: *weight-finite less-top*[*symmetric*])
    finally show *?thesis* **using** *Pair* **by** *simp*
  qed

have *chainD*: *Inf M* ∈ *F* **if** *M*: *M* ∈ *Chains leq* **and** *nempty*: *M* ≠ {} **for** *M*
proof −
  from *nempty* **obtain** *g0* **where** *g0*: *g0* ∈ *M* **by** *auto*
  have *g0-le-f*: *g0 e* ≤ *f e* **and** *g*: *current* Γ *g0* **and** *w0*: *wave* Γ *g0* **for** *e*
    using *Chains-FieldD*[*OF M g0*] **by**(*cases e*, *auto simp add*: *leq-def*)

  have *finite-OUT*: *d-OUT f x* ≠ ⊤ **for** *x* **using** *weight-finite*[*of x*]
    by(*rule neq-top-trans*)(*rule currentD-weight-OUT*[*OF f*])
  have *finite-IN*: *d-IN f x* ≠ ⊤ **for** *x* **using** *weight-finite*[*of x*]
    by(*rule neq-top-trans*)(*rule currentD-weight-IN*[*OF f*])

  from *M* **have** *M* ∈ *Chains* {(*g*, *g′*). *g′* ≤ *g*}
  by(*rule mono-Chains*[*THEN subsetD*, *rotated*])(*auto simp add*: *leq-def in-restrict-rel-iff*)
  then have *chain*: *Complete-Partial-Order.chain* (≥) *M* **by**(*rule Chains-into-chain*)
  hence *chain′*: *Complete-Partial-Order.chain* (≤) *M* **by**(*simp add*: *chain-dual*)

  have *countable′*: *countable* (*support-flow f*)
    using *current-support-flow*[*OF f*] **by**(*rule countable-subset*)(*rule countable*)

  have *OUT-M*: *d-OUT* (*Inf M*) *x* = (*INF g*∈*M*. *d-OUT g x*) **for** *x* **using** *chain′* *nempty countable′* - *finite-OUT*
    by(*rule d-OUT-Inf*)(*auto dest*!: *Chains-FieldD*[*OF M*] *simp add*: *leq-def*)
  have *IN-M*: *d-IN* (*Inf M*) *x* = (*INF g*∈*M*. *d-IN g x*) **for** *x* **using** *chain′ nempty countable′* - *finite-IN*
    by(*rule d-IN-Inf*)(*auto dest*!: *Chains-FieldD*[*OF M*] *simp add*: *leq-def*)

  have *c*: *current* Γ (*Inf M*) **using** *g*
  proof(*rule current-leI*)
    show (*Inf M*) *e* ≤ *g0 e* **for** *e* **using** *g0* **by**(*auto intro*: *INF-lower*)
    show *d-OUT* ($\bigsqcap$*M*) *x* ≤ *d-IN* ($\bigsqcap$*M*) *x* **if** *x* ∉ *A* Γ **for** *x*
      by(*auto 4 4 simp add*: *IN-M OUT-M leq-def intro*!: *INF-mono dest*: *Chains-FieldD*[*OF M*] *intro*: *currentD-OUT-IN*[*OF* - *that*])
  qed

  have *INF-le-f*: *Inf M e* ≤ *f e* **for** *e* **using** *g0* **by**(*auto intro*!: *INF-lower2 g0-le-f*)
  have *eq*: $\mathcal{E}$ (*TER* (*Inf M*)) = $\mathcal{E}$ (*TER f*) **using** *f w INF-le-f*
  proof(*rule essential-eq-leI*; *intro subsetI*)
    fix *x*
    assume *x*: *x* ∈ $\mathcal{E}$ (*TER f*)
    hence *x* ∈ *SINK* (*Inf M*) **using** *d-OUT-mono*[*of Inf M x f*, *OF INF-le-f*]
      by(*auto simp add*: *SINK.simps*)
    moreover from *x* **have** *x* ∈ *SAT* Γ *g* **if** *g* ∈ *M* **for** *g* **using** *Chains-FieldD*[*OF M that*] **by**(*auto simp add*: *leq-def*)
    hence *x* ∈ *SAT* Γ (*Inf M*) **by**(*auto simp add*: *SAT.simps IN-M intro*!: *INF-greatest*)
    ultimately show *x* ∈ *TER* (*Inf M*) **by** *auto*
  qed

  have *w′*: *wave* Γ (*Inf M*)

proof
have *separating* Γ ($\mathcal{E}$ (*TER f*)) **by**(*rule separating-essential*)(*rule waveD-separating*[*OF w*])
then show *separating* Γ (*TER* (*Inf M*)) **unfolding** *eq*[*symmetric*] **by**(*rule separating-weakening*) *auto*

fix *x*
assume $x \notin RF$ (*TER* (*Inf M*))
hence $x \notin RF$ ($\mathcal{E}$ (*TER* (*Inf M*))) **unfolding** *RF-essential* **.**
hence $x \notin RF$ (*TER f*) **unfolding** *eq RF-essential* **.**
hence *d-OUT f x = 0* **by**(*rule waveD-OUT*[*OF w*])
with *d-OUT-mono*[*of - x f*, *OF INF-le-f*]
show *d-OUT* (*Inf M*) *x = 0* **by** (*metis le-zero-eq*)
qed
from *c w′ INF-le-f eq* **show** *?thesis* **by** *simp*
qed

define *trim1*
where *trim1 g =*
(*if trimming g then g*
else let z = SOME z. $z \in RF^\circ$ (*TER g*) $\wedge$ $z \notin A\ \Gamma \wedge \neg$ *KIR g z*;
factor = d-OUT g z / d-IN g z
in (λ(*y*, *x*). (*if x = z then factor else 1*) $*$ *g* (*y*, *x*))) **for** *g*

have *increasing*: *trim1 g* $\leq$ *g* $\wedge$ *trim1 g* $\in$ *F* **if** *g* $\in$ *F* **for** *g*
proof(*cases trimming g*)
case *True*
thus *?thesis* **using** *that* **by**(*simp add*: *trim1-def*)
next
case *False*
let *?P* = $\lambda z.\ z \in RF^\circ$ (*TER g*) $\wedge$ $z \notin A\ \Gamma \wedge \neg$ *KIR g z*
define *z* **where** *z = Eps ?P*
from *that* **have** *g*: *current* Γ *g* **and** *w′*: *wave* Γ *g* **and** *le-f*: $\bigwedge e.\ g\ e \leq f\ e$
and $\mathcal{E}$: $\mathcal{E}$ (*TER g*) = $\mathcal{E}$ (*TER f*) **by**(*auto simp add*: *le-fun-def*)
{ with *False* **obtain** *z* **where** *z*: $z \in RF^\circ$ (*TER f*) **and** *A*: $z \notin A\ \Gamma$ **and** *neq*: *d-OUT g z* $\neq$ *d-IN g z*
by(*auto simp add*: *trimming.simps le-fun-def*)
from *z* **have** $z \in RF^\circ$ ($\mathcal{E}$ (*TER f*)) **unfolding** *roofed-circ-essential* **.**
with $\mathcal{E}$ *roofed-circ-essential*[*of* Γ *TER g*] **have** $z \in RF^\circ$ (*TER g*) **by** *simp*
with *A neq* **have** $\exists x.$ *?P x* **by** *auto* **}**
hence *?P z* **unfolding** *z-def* **by**(*rule someI-ex*)
hence *RF*: $z \in RF^\circ$ (*TER g*) **and** *A*: $z \notin A\ \Gamma$ **and** *neq*: *d-OUT g z* $\neq$ *d-IN g z* **by** *simp-all*
let *?factor = d-OUT g z / d-IN g z*
have *trim1* [*simp*]: *trim1 g* (*y*, *x*) = (*if x = z then ?factor else 1*) $*$ *g* (*y*, *x*) **for** *x y*
using *False* **by**(*auto simp add*: *trim1-def z-def Let-def*)

from *currentD-OUT-IN*[*OF g A*] *neq* **have** *less*: *d-OUT g z* < *d-IN g z* **by** *auto*

hence *?factor* $\leq$ *1* (**is** *?factor* $\leq$ -)
by (*auto intro*!: *divide-le-posI-ennreal simp*: *zero-less-iff-neq-zero*)
hence *le*$'$: *?factor* $*$ *g* (*y*, *x*) $\leq$ *1* $*$ *g* (*y*, *x*) **for** *y x*
by(*rule mult-right-mono*) *simp*
hence *le*: *trim1 g e* $\leq$ *g e* **for** *e* **by**(*cases e*)*simp*
moreover **{**
have *c*: *current* Γ (*trim1 g*) **using** *g le*
proof(*rule current-leI*)
fix *x*
assume *x*: *x* $\notin$ *A* Γ
have *d-OUT* (*trim1 g*) *x* $\leq$ *d-OUT g x* **unfolding** *d-OUT-def* **using** *le*$'$ **by**(*auto intro*: *nn-integral-mono*)
also have ... $\leq$ *d-IN* (*trim1 g*) *x*
proof(*cases x* = *z*)
case *True*
have *d-OUT g x* = *d-IN* (*trim1 g*) *x* **unfolding** *d-IN-def*
using *True currentD-weight-IN*[*OF g*, *of x*] *currentD-OUT-IN*[*OF g x*]
apply (*cases d-IN g x* = *0*)
apply(*auto simp add*: *nn-integral-divide nn-integral-cmult d-IN-def*[*symmetric*] *ennreal-divide-times*)
apply (*subst ennreal-divide-self*)
apply (*auto simp*: *less-top*[*symmetric*] *top-unique weight-finite*)
done
thus *?thesis* **by** *simp*
next
case *False*
have *d-OUT g x* $\leq$ *d-IN g x* **using** *x* **by**(*rule currentD-OUT-IN*[*OF g*])
also have ... $\leq$ *d-IN* (*trim1 g*) *x* **unfolding** *d-IN-def* **using** *False* **by**(*auto intro*!: *nn-integral-mono*)
finally show *?thesis* **.**
qed
finally show *d-OUT* (*trim1 g*) *x* $\leq$ *d-IN* (*trim1 g*) *x* **.**
qed
moreover have *le-f*: *trim1 g* $\leq$ *f* **using** *le le-f* **by**(*blast intro*: *le-funI order-trans*)
moreover have *eq*: $\mathcal{E}$ (*TER* (*trim1 g*)) = $\mathcal{E}$ (*TER f*) **unfolding** $\mathcal{E}$[*symmetric*] **using** *g w*$'$ *le*
proof(*rule essential-eq-leI*; *intro subsetI*)
fix *x*
assume *x*: *x* $\in$ $\mathcal{E}$ (*TER g*)
hence *x* $\in$ *SINK* (*trim1 g*) **using** *d-OUT-mono*[*of trim1 g x g*, *OF le*]
by(*auto simp add*: *SINK.simps*)
moreover from *x* **have** *x* $\neq$ *z* **using** *RF* **by**(*auto simp add*: *roofed-circ-def*)
hence *d-IN* (*trim1 g*) *x* = *d-IN g x* **unfolding** *d-IN-def* **by** *simp*
with ‹*x* $\in$ $\mathcal{E}$ (*TER g*)› **have** *x* $\in$ *SAT* Γ (*trim1 g*) **by**(*auto simp add*: *SAT.simps*)
ultimately show *x* $\in$ *TER* (*trim1 g*) **by** *auto*
qed
moreover have *wave* Γ (*trim1 g*)

proof
have *separating* Γ ($\mathcal{E}$ (*TER f*)) **by**(*rule separating-essential*)(*rule waveD-separating*[*OF w*])
then show *separating* Γ (*TER* (*trim1 g*)) **unfolding** *eq*[*symmetric*] **by**(*rule separating-weakening*) *auto*

fix *x*
assume $x \notin$ *RF* (*TER* (*trim1 g*))
hence $x \notin$ *RF* ($\mathcal{E}$ (*TER* (*trim1 g*))) **unfolding** *RF-essential* **.**
hence $x \notin$ *RF* (*TER f*) **unfolding** *eq RF-essential* **.**
hence *d-OUT f x* = *0* **by**(*rule waveD-OUT*[*OF w*])
with *d-OUT-mono*[*of - x f*, *OF le-f*[*THEN le-funD*]]
show *d-OUT* (*trim1 g*) *x* = *0* **by** (*metis le-zero-eq*)
qed
ultimately have *trim1 g* $\in$ *F* **by**(*simp add*: *F-def*) **}**
ultimately show *?thesis* **using** *that* **by**(*simp add*: *le-fun-def del*: *trim1*)
qed

have *bourbaki-witt-fixpoint Inf leq trim1* **using** *chainD increasing* **unfolding** *leq-def*
by(*intro bourbaki-witt-fixpoint-restrict-rel*)(*auto intro*: *Inf-greatest Inf-lower*)
then interpret *bourbaki-witt-fixpoint Inf leq trim1* **.**

have *f-Field*: *f* $\in$ *Field leq* **using** *f w* **by**(*simp add*: *leq-def*)

define *g* **where** *g* = *fixp-above f*

have *g* $\in$ *Field leq* **using** *f-Field* **unfolding** *g-def* **by**(*rule fixp-above-Field*)
hence *le-f*: $g \le f$
and *g*: *current* Γ *g*
and *w'*: *wave* Γ *g*
and *TER*: $\mathcal{E}$ (*TER g*) = $\mathcal{E}$ (*TER f*) **by**(*auto simp add*: *leq-def intro*: *le-funI*)

have *trimming g*
proof(*rule ccontr*)
let *?P* = λx. $x \in$ *RF*$^\circ$ (*TER g*) $\wedge$ $x \notin$ *A* Γ $\wedge$ $\neg$ *KIR g x*
define *x* **where** *x* = *Eps ?P*
assume *False*: $\neg$ *?thesis*
hence $\exists$ *x*. *?P x* **using** *le-f g w' TER*
by(*auto simp add*: *trimming.simps roofed-circ-essential*[*of* Γ *TER g*, *symmetric*] *roofed-circ-essential*[*of* Γ *TER f*, *symmetric*])
hence *?P x* **unfolding** *x-def* **by**(*rule someI-ex*)
hence *x*: $x \in$ *RF*$^\circ$ (*TER g*) **and** *A*: $x \notin$ *A* Γ **and** *neq*: *d-OUT g x* $\neq$ *d-IN g x* **by** *simp-all*
from *neq* **have** $\exists$ *y*. *edge* Γ *y x* $\wedge$ *g* (*y*, *x*) > *0*
proof(*rule contrapos-np*)
assume $\neg$ *?thesis*
hence *d-IN g x* = *0* **using** *currentD-outside*[*OF g*, *of - x*]
by(*force simp add*: *d-IN-def nn-integral-0-iff-AE AE-count-space not-less*)

with *currentD-OUT-IN*[*OF g A*] **show** *KIR g x* **by** *simp*
qed
then obtain *y* **where** *y*: *edge* Γ *y x* **and** *gr0*: *g* (*y*, *x*) > *0* **by** *blast*

have [*simp*]: *g* (*y*, *x*) ≠ ⊤
proof −
have *g* (*y*, *x*) ≤ *d-OUT g y* **by** (*rule d-OUT-ge-point*)
also have ... ≤ *weight* Γ *y* **by**(*rule currentD-weight-OUT*[*OF g*])
also have ... < ⊤ **by**(*simp add*: *weight-finite less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

from *neq* **have** *factor*: *d-OUT g x* / *d-IN g x* ≠ *1*
by (*simp add*: *divide-eq-1-ennreal*)

have *trim1 g* (*y*, *x*) = *g* (*y*, *x*) ∗ (*d-OUT g x* / *d-IN g x*)
by(*clarsimp simp add*: *False trim1-def Let-def x-def*[*symmetric*] *mult.commute*)
moreover have ... ≠ *g* (*y*, *x*) ∗ *1* **unfolding** *ennreal-mult-cancel-left* **using** *gr0 factor* **by** *auto*
ultimately have *trim1 g* (*y*, *x*) ≠ *g* (*y*, *x*) **by** *auto*
hence *trim1 g* ≠ *g* **by**(*auto simp add*: *fun-eq-iff*)
moreover have *trim1 g* = *g* **using** *f-Field* **unfolding** *g-def* **by**(*rule fixp-above-unfold*[*symmetric*])
ultimately show *False* **by** *contradiction*
qed
then show *?thesis* **by** *blast*
qed

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*)
proof(*rule set-eqI*)
show *x* ∈ $\mathcal{E}$ (*TER f*) ⟷ *x* ∈ $\mathcal{E}$ (*TER g*) **for** *x*
proof(*cases x* ∈ *A* Γ)
case *False*
thus *?thesis* **using** *trimmingD-*$\mathcal{E}$[*OF trimming*] **by** *blast*
next
case *True*
show *?thesis*
proof
assume *x*: *x* ∈ $\mathcal{E}$ (*TER f*)
hence *x* ∈ *TER g* **using** *d-OUT-mono*[*of g x f*, *OF trimmingD-le*[*OF trimming*]] *True*
by(*simp add*: *SINK.simps SAT.A*)
moreover from *x* **have** *essential* Γ (*B* Γ) (*TER f*) *x* **by** *simp*
then obtain *p y* **where** *p*: *path* Γ *x p y* **and** *y*: *y* ∈ *B* Γ
and *bypass*: ⋀*z*. *z* ∈ *set p* ⟹ *z* ∉ *RF* (*TER f*) **by**(*rule essentialE-RF*)

blast
  from p y **have** *essential* Γ (B Γ) ($\mathcal{E}$ (*TER* g)) x
  proof(*rule essentialI*)
    fix z
    assume $z \in$ *set* p
    hence z: $z \notin$ *RF* (*TER* f) **by**(*rule bypass*)
    with *waveD-separating*[*OF* w, *THEN separating-RF-A*] **have** $z \notin A$ Γ **by** *blast*
    with z **have** $z \notin \mathcal{E}$ (*TER* g) **using** *trimmingD-*$\mathcal{E}$[*OF trimming*] **by**(*auto intro*: *roofed-greaterI*)
    thus $z = x \vee z \notin \mathcal{E}$ (*TER* g) **..**
  qed
  ultimately show $x \in \mathcal{E}$ (*TER* g) **unfolding** *essential-*$\mathcal{E}$ **by** *simp*
next
  assume $x \in \mathcal{E}$ (*TER* g)
  then obtain p y **where** p: *path* Γ x p y **and** y: $y \in B$ Γ
    and *bypass*: $\bigwedge z.$ $z \in$ *set* $p \Longrightarrow z \notin$ *RF* (*TER* g) **by**(*rule* $\mathcal{E}$*-E-RF*) *blast*
  have z: $z \notin \mathcal{E}$ (*TER* f) **if** $z \in$ *set* p **for** z
  proof −
    from *that* **have** z: $z \notin$ *RF* (*TER* g) **by**(*rule bypass*)
    with *waveD-separating*[*OF trimmingD-wave*[*OF trimming*], *THEN separating-RF-A*] **have** $z \notin A$ Γ **by** *blast*
    with z **show** $z \notin \mathcal{E}$ (*TER* f) **using** *trimmingD-*$\mathcal{E}$[*OF trimming*] **by**(*auto intro*: *roofed-greaterI*)
  qed
  then have *essential* Γ (B Γ) ($\mathcal{E}$ (*TER* f)) x **by**(*intro essentialI*[*OF* p y]) *auto*
  moreover have $x \in$ *TER* f
    using *waveD-separating*[*THEN separating-essential*, *THEN separatingD*, *OF* w p *True* y] z
    by *auto*
  ultimately show $x \in \mathcal{E}$ (*TER* f) **unfolding** *essential-*$\mathcal{E}$ **by** *simp*
qed
qed
qed

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 Γ − A Γ 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* = λx y. *edge* Γ x y $\wedge$ $x \notin$ *roofed-circ* Γ (*TER*$_\Gamma$ f) $\wedge$ $y \notin$ *roofed* Γ (*TER*$_\Gamma$ f),
    weight = λx. *if* $x \in RF^\circ{}_\Gamma$ (*TER*$_\Gamma$ f) $\vee$ $x \in$ *TER*$_\Gamma$ $f \cap B$ Γ *then 0 else weight* Γ x,
    $A = \mathcal{E}_\Gamma$ (*TER*$_\Gamma$ f) − (B Γ − A Γ),
    $B = B$ Γ,

$\ldots = \mathit{web.more}\ \Gamma\|)$

lemma *quotient-web-sel* [*simp*]:
fixes Γ (**structure**) **shows**
edge (*quotient-web* Γ *f*) *x y* $\longleftrightarrow$ *edge* Γ *x y* $\wedge$ $x \notin RF^{\circ}$ (*TER f*) $\wedge$ $y \notin RF$ (*TER f*)
weight (*quotient-web* Γ *f*) $x =$ (*if* $x \in RF^{\circ}$ (*TER f*) $\vee$ $x \in TER_{\Gamma}\ f \cap B\ \Gamma$ *then 0 else weight* Γ *x*)
A (*quotient-web* Γ *f*) $= \mathcal{E}$ (*TER f*)$-$ ($B\ \Gamma - A\ \Gamma$)
B (*quotient-web* Γ *f*) $= B\ \Gamma$
web.more (*quotient-web* Γ *f*) $=$ *web.more* Γ
by(*simp-all add*: *quotient-web-def*)

lemma *vertex-quotient-webD*: **fixes** Γ (**structure**) **shows**
vertex (*quotient-web* Γ *f*) $x \Longrightarrow$ *vertex* Γ $x \wedge x \notin RF^{\circ}$ (*TER f*)
by(*auto simp add*: *vertex-def roofed-circ-def*)

lemma *path-quotient-web*:
fixes Γ (**structure**)
assumes *path* Γ *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-web* Γ *f*) *x p y*
using *assms* **by**(*induction*)(*auto intro*: *rtrancl-path.intros simp add*: *roofed-circ-def*)

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

abbreviation *restrict-curr* :: $'v$ *current* $\Rightarrow$ ($'v$, $'more$) *web-scheme* $\Rightarrow$ $'v$ *current* $\Rightarrow$ $'v$ *current* (‹- ↿ - '/ -› [*100*, *0*, *100*] *100*)
where *restrict-curr* *g* Γ *f* $\equiv$ *restrict-current* Γ *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*)
by(*simp add*: *restrict-current-def*)

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

lemma *d-IN-restrict-current-outside*: **fixes** Γ (**structure**) **shows**
$x \in RF$ (*TER f*) $\Longrightarrow$ *d-IN* ($g \upharpoonleft \Gamma\ /\ f$) $x = 0$
by(*simp add*: *d-IN-def*)

lemma *restrict-current-le*: ($g \upharpoonleft \Gamma\ /\ f$) $e \leq g\ e$
by(*cases e*)(*clarsimp split*: *split-indicator*)

lemma *d-OUT-restrict-current-le*: *d-OUT* $(g \upharpoonright \Gamma \ / \ f)$ $x \leq$ *d-OUT* g x
unfolding *d-OUT-def* **by**(*rule nn-integral-mono*, *simp split*: *split-indicator*)

lemma *d-IN-restrict-current-le*: *d-IN* $(g \upharpoonright \Gamma \ / \ f)$ $x \leq$ *d-IN* g x
unfolding *d-IN-def* **by**(*rule nn-integral-mono*, *simp split*: *split-indicator*)

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 −
  {
    fix y
    assume *y*: $y \in RF^{\circ}$ (*TER* f)
    have g $(y, x) = 0$
    proof(*cases edge* Γ y x)
      case *True*
      from y **have** y': $y \in$ *RF* (*TER* f) **and** *essential*: $y \notin \mathcal{E}$ (*TER* f) **by**(*simp-all add*: *roofed-circ-def*)
      moreover from x **obtain** p z **where** *z*: $z \in B$ Γ **and** *p*: *path* Γ x p z
        and *bypass*: $\bigwedge z.\ z \in$ *set* $p \Longrightarrow z \notin$ *TER* f $x \notin$ *TER* f
        by(*clarsimp simp add*: *roofed-def*)
      from *roofedD*[*OF* y' *rtrancl-path.step*, *OF True* p z] *bypass* **have** $x \in$ *TER* f $\vee$ $y \in$ *TER* f **by** *auto*
      with *roofed-greater*[*THEN subsetD*, *of* x *TER* f Γ] x **have** $x \notin$ *TER* f $y \in$ *TER* f **by** *auto*
      with *essential bypass* **have** *False*
        by(*auto dest*!: *not-essentialD*[*OF* - *rtrancl-path.step*, *OF* - *True* p z])
      thus *?thesis* **..**
    qed(*simp add*: *currentD-outside*[*OF* g]) **}**
  then show *?thesis* **unfolding** *d-IN-def*
    using x **by**(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
qed

lemma *restrict-current-IN-A*:
  $a \in A$ (*quotient-web* Γ f) $\Longrightarrow$ *d-IN* $(g \upharpoonright \Gamma \ / \ f)$ $a = 0$
by(*simp add*: *d-IN-restrict-current-outside roofed-greaterI*)

lemma *restrict-current-nonneg*: $0 \leq g$ $e \Longrightarrow 0 \leq (g \upharpoonright \Gamma \ / \ f)$ e
by(*cases* e) *simp*

lemma *in-SINK-restrict-current*: $x \in$ *SINK* $g \Longrightarrow x \in$ *SINK* $(g \upharpoonright \Gamma \ / \ f)$
using *d-OUT-restrict-current-le*[*of* Γ f g x]
by(*simp add*: *SINK.simps*)

lemma *SAT-restrict-current*:
  fixes Γ (**structure**)

assumes *f*: *current* Γ *f*
and *g*: *current* Γ *g*
shows *SAT* (*quotient-web* Γ *f*) (*g* ↿ Γ / *f*) = *RF* (*TER f*) ∪ (*SAT* Γ *g* − *A* Γ)
(**is** *SAT ?*Γ *?g* = *?rhs*)
proof(*intro set-eqI iffI*; (*elim UnE DiffE*)*?*)
show $x \in$ *?rhs* **if** $x \in$ *SAT ?*Γ *?g* **for** *x* **using** *that*
proof *cases*
case *IN*
thus *?thesis* **using** *currentD-weight-OUT*[*OF f*, *of x*]
by(*cases* $x \in$ *RF* (*TER f*))(*auto simp add*: *d-IN-restrict-current-outside roofed-circ-def restrict-current-IN-not-RF*[*OF g*] *SAT.IN currentD-IN*[*OF g*] *roofed-greaterI SAT.A SINK.simps RF-in-B split*: *if-split-asm intro*: *essentialI*[*OF rtrancl-path.base*])
qed(*simp add*: *roofed-greaterI*)
show $x \in$ *SAT ?*Γ *?g* **if** $x \in$ *RF* (*TER f*) **for** *x* **using** *that*
by(*auto simp add*: *SAT.simps roofed-circ-def d-IN-restrict-current-outside*)
show $x \in$ *SAT ?*Γ *?g* **if** $x \in$ *SAT* Γ *g* $x \notin$ *A* Γ **for** *x* **using** *that*
by(*auto simp add*: *SAT.simps roofed-circ-def d-IN-restrict-current-outside restrict-current-IN-not-RF*[*OF g*])
qed

lemma *current-restrict-current*:
fixes Γ (**structure**)
assumes *w*: *wave* Γ *f*
and *g*: *current* Γ *g*
shows *current* (*quotient-web* Γ *f*) (*g* ↿ Γ / *f*) (**is** *current ?*Γ *?g*)
proof
show *d-OUT ?g x* ≤ *weight ?*Γ *x* **for** *x*
using *d-OUT-restrict-current-le*[*of* Γ *f g x*] *currentD-weight-OUT*[*OF g*, *of x*] *currentD-OUT*[*OF g*, *of x*]
by(*auto simp add*: *d-OUT-restrict-current-outside*)
show *d-IN ?g x* ≤ *weight ?*Γ *x* **for** *x*
using *d-IN-restrict-current-le*[*of* Γ *f g x*] *currentD-weight-IN*[*OF g*, *of x*]
by(*auto simp add*: *d-IN-restrict-current-outside roofed-circ-def*)
(*subst d-IN-restrict-current-outside*[*of x* Γ *f g*]; *simp add*: *roofed-greaterI*)

fix *x*
assume $x \notin$ *A ?*Γ
hence *x*: $x \notin \mathcal{E}$ (*TER f*) − *B* Γ **by** *simp*
show *d-OUT ?g x* ≤ *d-IN ?g x*
proof(*cases* $x \in$ *RF* (*TER f*))
case *True*
with *x* **have** $x \in RF^{\circ}$ (*TER f*) ∪ *B* Γ **by**(*simp add*: *roofed-circ-def*)
with *True* **show** *?thesis* **using** *currentD-OUT*[*OF g*, *of x*] *d-OUT-restrict-current-le*[*of* Γ *f g x*]
by(*auto simp add*: *d-OUT-restrict-current-outside d-IN-restrict-current-outside*)
next
case *False*
then obtain *p z* **where** *z*: $z \in$ *B* Γ **and** *p*: *path* Γ *x p z*
and *bypass*: ⋀*z*. $z \in$ *set p* ⟹ $z \notin$ *TER f x* $\notin$ *TER f*

by(*clarsimp simp add*: *roofed-def*)
from *g False* **have** *d-IN ?g x = d-IN g x* **by**(*rule restrict-current-IN-not-RF*)
moreover have *d-OUT ?g x* $\leq$ *d-OUT g x*
by(*rule d-OUT-mono restrict-current-le*)+
moreover have $x \notin A\ \Gamma$
using *separatingD*[*OF waveD-separating*[*OF w*] *p - z*] *bypass* **by** *blast*
note *currentD-OUT-IN*[*OF g this*]
ultimately show *?thesis* **by** *simp*
qed
next
show *d-IN ?g a = 0* **if** $a \in A$?Γ **for** *a* **using** *that* **by**(*rule restrict-current-IN-A*)
show *d-OUT ?g b = 0* **if** $b \in B$?Γ **for** *b*
using *d-OUT-restrict-current-le*[*of* Γ *f g b*] *currentD-OUT*[*OF g*, *of b*] *that* **by** *simp*
show *?g e = 0* **if** $e \notin \mathbf{E}_{?\Gamma}$ **for** *e* **using** *that currentD-outside'*[*OF g*, *of e*]
by(*cases e*)(*auto split*: *split-indicator*)
qed

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* $\upharpoonright$ Γ / *f*) (**is** - $\subseteq$ *?TER* **is** - $\subseteq$ $TER_{?\Gamma}$ *?g*)
proof
fix *x*
assume *x*: $x \in$ *TER g*
hence $x \in$ *SINK ?g* **by**(*simp add*: *in-SINK-restrict-current*)
moreover have $x \in$ *RF* (*TER f*) **if** $x \in A\ \Gamma$
using *waveD-separating*[*OF w*, *THEN separatingD*, *OF - that*] **by**(*rule roofedI*)
then have $x \in$ *SAT* ?Γ *?g* **using** *SAT-restrict-current*[*OF f g*] *x* **by** *auto*
ultimately show $x \in$ *?TER* **by** *simp*
qed

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* $\upharpoonright$ Γ / *f*) (**is** *wave* ?Γ *?g*)
proof
show *separating* ?Γ ($TER_{?\Gamma}$ *?g*) (**is** *separating - ?TER*)
proof
fix *x y p*
assume $x \in A$?Γ $y \in B$?Γ **and** *p*: *path* ?Γ *x p y*
hence *x*: $x \in \mathcal{E}$ (*TER f*) **and** *y*: $y \in B\ \Gamma$ **and** *SAT*: $x \in$ *SAT* ?Γ *?g* **by**(*simp-all add*: *SAT.A*)

from *p* **have** *p'*: *path* Γ *x p y* **by**(*rule rtrancl-path-mono*) *simp*

{ assume *x* ∉ *?TER*
hence *x* ∉ *SINK ?g* **using** *SAT* **by**(*simp*)
hence *x* ∉ *SINK g* **using** *d-OUT-restrict-current-le*[*of* Γ *f g x*]
by(*auto simp add*: *SINK.simps*)
hence *x* ∈ *RF* (*TER g*) **using** *waveD-OUT*[*OF w'*] **by**(*auto simp add*: *SINK.simps*)
from *roofedD*[*OF this p' y*] ‹*x* ∉ *SINK g*› **have** (∃ *z*∈*set p*. *z* ∈ *?TER*)
using *TER-restrict-current*[*OF f w g*] **by** *blast* **}**
then show (∃ *z*∈*set p*. *z* ∈ *?TER*) ∨ *x* ∈ *?TER* **by** *blast*
qed

fix *x*
assume *x* ∉ $RF_{?\Gamma}$ *?TER*
hence *x* ∉ *RF* (*TER g*)
proof(*rule contrapos-nn*)
assume ∗: *x* ∈ *RF* (*TER g*)
show *x* ∈ $RF_{?\Gamma}$ *?TER*
proof
fix *p y*
assume *path ?*Γ *x p y y* ∈ *B ?*Γ
hence *path* Γ *x p y y* ∈ *B* Γ **by**(*auto elim*: *rtrancl-path-mono*)
from *roofedD*[*OF* ∗ *this*] **show** (∃ *z*∈*set p*. *z* ∈ *?TER*) ∨ *x* ∈ *?TER*
using *TER-restrict-current*[*OF f w g*] **by** *blast*
qed
qed
with *w'* **have** *d-OUT g x* = *0* **by**(*rule waveD-OUT*)
with *d-OUT-restrict-current-le*[*of* Γ *f g x*]
show *d-OUT ?g x* = *0* **by** *simp*
qed

definition *plus-current* :: *'v current* ⇒ *'v current* ⇒ *'v current*
where *plus-current f g* = (λ*e*. *f e* + *g e*)

lemma *plus-current-simps* [*simp*]: *plus-current f g e* = *f e* + *g e*
by(*simp add*: *plus-current-def*)

lemma *plus-zero-current* [*simp*]: *plus-current f zero-current* = *f*
by(*simp add*: *fun-eq-iff*)

lemma *support-flow-plus-current*: *support-flow* (*plus-current f g*) ⊆ *support-flow f* ∪ *support-flow g*
by(*clarsimp simp add*: *support-flow.simps*)

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 ∈ RF° (TER f) then d-OUT f x else d-OUT g x) (**is** d-OUT ?g - = -)
proof −
  have d-OUT ?g x = d-OUT f x + d-OUT g x **unfolding** plus-current-def
    by(subst d-OUT-add) simp-all
  also have … = (if x ∈ RF° (TER f) then d-OUT f x else d-OUT g x)
  proof(cases x ∈ RF° (TER f))
    case True
    hence d-OUT g x = 0 **by**(intro currentD-outside-OUT[OF g])(auto dest: vertex-quotient-webD)
    thus ?thesis **using** True **by** simp
  next
    case False
    hence d-OUT f x = 0 **by**(auto simp add: roofed-circ-def SINK.simps dest: waveD-OUT[OF w])
    with False **show** ?thesis **by** simp
  qed
  finally show ?thesis **.**
qed

lemma IN-plus-current: d-IN (plus-current f g) x = (if x ∈ RF (TER f) then d-IN f x else d-IN g x) (**is** d-IN ?g - = -)
proof −
  have d-IN ?g x = d-IN f x + d-IN g x **unfolding** plus-current-def
    by(subst d-IN-add) simp-all
  also consider (RF) x ∈ RF (TER f) − (B Γ − A Γ) | (B) x ∈ RF (TER f) x ∈ B Γ − A Γ | (beyond) x ∉ RF (TER f) **by** blast
  then have d-IN f x + d-IN g x = (if x ∈ RF (TER f) then d-IN f x else d-IN g x)
  proof(cases)
    case RF
    hence d-IN g x = 0
     by(cases x ∈ ℰ (TER f))(auto intro: currentD-outside-IN[OF g] currentD-IN[OF g] dest!: vertex-quotient-webD simp add: roofed-circ-def)
    thus ?thesis **using** RF **by** simp
  next
    case B
    hence d-IN g x = 0 **using** currentD-outside-IN[OF g, of x] currentD-weight-IN[OF g, of x]
      by(auto dest: vertex-quotient-webD simp add: roofed-circ-def)
    with B **show** ?thesis **by** simp
  next
    case beyond
    from f w beyond **have** d-IN f x = 0 **by**(rule wave-not-RF-IN-zero)
    with beyond **show** ?thesis **by** simp
  qed

finally show *?thesis* **.**
qed

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(*cases* $x \in \mathcal{E}$ (*TER f*) $-$ (*B* Γ $-$ *A* Γ))
case *True*
with *x* **show** *?thesis* **using** *currentD-IN*[*OF g*, *of x*]
by(*fastforce intro*: *roofed-greaterI SAT.intros simp add*: *SINK.simps OUT-plus-current IN-plus-current elim*!: *SAT.cases*)
next
case $*$: *False*
have $x \in$ *SAT* Γ *?g*
proof(*cases* $x \in B\ \Gamma - A\ \Gamma$)
case *False*
with *x RF* $*$ **have** *weight* Γ $x \leq$ *d-IN g x*
by(*auto elim*!: *SAT.cases split*: *if-split-asm simp add*: *essential-BI*)
also have $\ldots \leq$ *d-IN ?g x* **unfolding** *plus-current-def* **by**(*intro d-IN-mono*) *simp*
finally show *?thesis* **..**
next
case *True*
with $*$ *x* **have** *weight* Γ $x \leq$ *d-IN ?g x* **using** *currentD-OUT*[*OF f*, *of x*]
by(*auto simp add*: *IN-plus-current RF-in-B SINK.simps roofed-circ-def elim*!: *SAT.cases split*: *if-split-asm*)
thus *?thesis* **..**
qed
moreover have $x \in$ *SINK ?g* **using** *x* **by**(*simp add*: *SINK.simps OUT-plus-current RF*)
ultimately show *?thesis* **by** *simp*
qed

lemma *current-plus-current*: *current* Γ (*plus-current f g*) (**is** *current* - *?g*)
proof
show *d-OUT ?g x* $\leq$ *weight* Γ *x* **for** *x*
using *currentD-weight-OUT*[*OF g*, *of x*] *currentD-weight-OUT*[*OF f*, *of x*]
by(*auto simp add*: *OUT-plus-current split*: *if-split-asm elim*: *order-trans*)
show *d-IN ?g x* $\leq$ *weight* Γ *x* **for** *x*
using *currentD-weight-IN*[*OF f*, *of x*] *currentD-weight-IN*[*OF g*, *of x*]
by(*auto simp add*: *IN-plus-current roofed-circ-def split*: *if-split-asm elim*: *order-trans*)
show *d-OUT ?g x* $\leq$ *d-IN ?g x* **if** $x \notin A\ \Gamma$ **for** *x*
proof(*cases* $x \in \mathcal{E}$ (*TER f*))
case *False*
thus *?thesis*
using *currentD-OUT-IN*[*OF f that*] *currentD-OUT-IN*[*OF g*, *of x*] *that*
by(*auto simp add*: *OUT-plus-current IN-plus-current roofed-circ-def SINK.simps*)

next
case *True*
with *that* **have** $d\text{-}OUT\ f\ x = 0\ weight\ \Gamma\ x \leq d\text{-}IN\ f\ x$
by(*auto simp add*: *SINK.simps elim*: *SAT.cases*)
thus *?thesis* **using** *that True currentD-OUT-IN*[*OF g*, *of x*] *currentD-weight-OUT*[*OF g*, *of x*]
by(*auto simp add*: *OUT-plus-current IN-plus-current roofed-circ-def intro*: *roofed-greaterI split*: *if-split-asm*)
qed
show $d\text{-}IN\ ?g\ a = 0$ **if** $a \in A\ \Gamma$ **for** a
using *wave-A-in-RF*[*OF w that*] *currentD-IN*[*OF f that*] **by**(*simp add*: *IN-plus-current*)
show $d\text{-}OUT\ ?g\ b = 0$ **if** $b \in B\ \Gamma$ **for** b
using *that currentD-OUT*[*OF f that*] *currentD-OUT*[*OF g*, *of b*] *that*
by(*auto simp add*: *OUT-plus-current SINK.simps roofed-circ-def intro*: *roofed-greaterI*)
show $?g\ e = 0$ **if** $e \notin \mathbf{E}$ **for** e **using** *currentD-outside'*[*OF f*, *of e*] *currentD-outside'*[*OF g*, *of e*] *that*
by(*cases e*) *auto*
qed

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* $\Gamma\ x\ p\ y$
shows $(\exists z \in set\ p.\ z \in TER\ (plus\text{-}current\ f\ g)) \vee x \in TER\ (plus\text{-}current\ f\ g)$ (**is** $- \vee - \in TER\ ?g$)
proof −
from x **have** $x \in RF\ (\mathcal{E}\ (TER\ f))$ **unfolding** *RF-essential* **.**
from *roofedD*[*OF this p y*] **have** $\exists z \in set\ (x\ \#\ p).\ z \in \mathcal{E}\ (TER\ f)$ **by** *auto*
from *split-list-last-prop*[*OF this*] **obtain** *ys z zs*
where *decomp*: $x\ \#\ p = ys\ @\ z\ \#\ zs$ **and** z: $z \in \mathcal{E}\ (TER\ f)$
and *outside*: $\bigwedge z.\ z \in set\ zs \Longrightarrow z \notin \mathcal{E}\ (TER\ f)$ **by** *auto*
have *zs*: *path* Γ *z zs y* **using** *decomp p*
by(*cases ys*)(*auto elim*: *rtrancl-path-appendE*)
moreover have $z \notin RF^{\circ}\ (TER\ f)$ **using** z **by**(*simp add*: *roofed-circ-def*)
moreover have *RF*: $z' \notin RF\ (TER\ f)$ **if** $z' \in set\ zs$ **for** z'
proof
assume $z' \in RF\ (TER\ f)$
hence z': $z' \in RF\ (\mathcal{E}\ (TER\ f))$ **by**(*simp only*: *RF-essential*)
from *split-list*[*OF that*] **obtain** $ys'\ zs'$ **where** $decomp'$: $zs = ys'\ @\ z'\ \#\ zs'$ **by** *blast*
with *zs* **have** *path* $\Gamma\ z'\ zs'\ y$ **by**(*auto elim*: *rtrancl-path-appendE*)
from *roofedD*[*OF* z' *this y*] *outside* $decomp'$ **show** *False* **by** *auto*
qed
ultimately have p': *path* (*quotient-web* Γ *f*) *z zs y* **by**(*rule path-quotient-web*)
show *?thesis*
proof(*cases* $z \in B\ \Gamma - A\ \Gamma$)
case *False*

with *separatingD*[*OF waveD-separating*[*OF w'*] *p'*] *z y*
obtain *z'* **where** *z'*: $z' \in set\ (z \# zs)$ **and** *TER*: $z' \in TER_{quotient\text{-}web\ \Gamma\ f}\ g$ **by** *auto*
hence $z' \in TER\ ?g$ **using** *in-TER-plus-current*[*of z'*] *RF*[*of z'*] ‹$z \notin RF^{\circ}$ (*TER f*)› **by**(*auto simp add*: *roofed-circ-def*)
with *decomp z'* **show** *?thesis* **by**(*cases ys*) *auto*
next
case *True*
hence $z \in TER\ ?g$ **using** *currentD-OUT*[*OF current-plus-current*, *of z*] *z*
by(*auto simp add*: *SINK.simps SAT.simps IN-plus-current intro*: *roofed-greaterI*)
then show *?thesis* **using** *decomp* **by**(*cases ys*) *auto*
qed
qed

lemma *wave-plus-current*: *wave* Γ (*plus-current f g*) (**is** *wave - ?g*)
proof
let *?*Γ = *quotient-web* Γ *f*
let $?TER = TER_{?\Gamma}$

show *separating* Γ (*TER ?g*) **using** *separating-TER-plus-current*[*OF wave-A-in-RF*[*OF w*]] **by**(*rule separating*)

fix *x*
assume *x*: $x \notin RF\ (TER\ ?g)$
hence $x \notin RF\ (TER\ f)$ **by**(*rule contrapos-nn*)(*rule roofedI*, *rule separating-TER-plus-current*)
hence ∗: $x \notin RF^{\circ}\ (TER\ f)$ **by**(*simp add*: *roofed-circ-def*)
moreover have $x \notin RF_{?\Gamma}\ (?TER\ g)$
proof
assume *RF'*: $x \in RF_{?\Gamma}\ (?TER\ g)$
from *x* **obtain** *p y* **where** *y*: $y \in B\ \Gamma$ **and** *p*: *path* Γ *x p y*
and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin TER\ ?g$ **and** *x'*: $x \notin TER\ ?g$
by(*auto simp add*: *roofed-def*)
have *RF*: $z \notin RF\ (TER\ f)$ **if** $z \in set\ p$ **for** *z*
proof
assume *z*: $z \in RF\ (TER\ f)$
from *split-list*[*OF that*] **obtain** *ys' zs'* **where** *decomp*: $p = ys'\ @\ z \#\ zs'$ **by** *blast*
with *p* **have** *path* Γ *z zs' y* **by**(*auto elim*: *rtrancl-path-appendE*)
from *separating-TER-plus-current*[*OF z y this*] *decomp bypass* **show** *False* **by** *auto*
qed
with *p* **have** *path ?*Γ *x p y* **using** ∗
by(*induction*)(*auto intro*: *rtrancl-path.intros simp add*: *roofed-circ-def*)
from *roofedD*[*OF RF' this*] *y* **consider** (*x*) $x \in ?TER\ g$ | (*z*) *z* **where** $z \in set\ p\ z \in ?TER\ g$ **by** *auto*
then show *False*
proof(*cases*)
case *x*
with ∗ **have** $x \in TER\ ?g$ **by**(*rule in-TER-plus-current*)

with x' **show** *False* **by** *contradiction*
next
case $(z\ z)$
from $z(1)$ **have** $z \notin RF\ (TER\ f)$ **by**(*rule RF*)
hence $z \notin RF^{\circ}\ (TER\ f)$ **by**(*simp add*: *roofed-circ-def*)
hence $z \in TER\ ?g$ **using** $z(2)$ **by**(*rule in-TER-plus-current*)
moreover from $z(1)$ **have** $z \notin TER\ ?g$ **by**(*rule bypass*)
ultimately show *False* **by** *contradiction*
qed
qed
with w' **have** $d\text{-}OUT\ g\ x = 0$ **by**(*rule waveD-OUT*)
ultimately show $d\text{-}OUT\ ?g\ x = 0$ **by**(*simp add*: *OUT-plus-current*)
qed

end

end

lemma *loose-quotient-web*:
fixes Γ :: $('v, 'more)$ *web-scheme* (**structure**)
assumes *weight-finite*: $\bigwedge x.\ weight\ \Gamma\ x \neq \top$
and f: *current* Γ f
and w: *wave* Γ f
and *maximal*: $\bigwedge w.\ [\![\ current\ \Gamma\ w;\ wave\ \Gamma\ w;\ f \leq w\ ]\!] \Longrightarrow f = w$
shows *loose* (*quotient-web* Γ f) (**is** *loose* $?\Gamma$)
proof
fix g
assume g: *current* $?\Gamma$ g **and** w': *wave* $?\Gamma$ g
let $?g = plus\text{-}current\ f\ g$
from $f\ w\ g$ **have** *current* Γ $?g$ *wave* Γ $?g$ **by**(*rule current-plus-current wave-plus-current*)+ (*rule* w')
moreover have $f \leq ?g$ **by**(*clarsimp simp add*: *le-fun-def add-eq-0-iff-both-eq-0*)
ultimately have *eq*: $f = ?g$ **by**(*rule maximal*)
have $g\ e = 0$ **for** e
proof(*cases e*)
case (*Pair x y*)
have $f\ e \leq d\text{-}OUT\ f\ x$ **unfolding** *Pair* **by** (*rule d-OUT-ge-point*)
also have $\ldots \leq weight\ \Gamma\ x$ **by**(*rule currentD-weight-OUT*[*OF f*])
also have $\ldots < \top$ **by**(*simp add*: *weight-finite less-top*[*symmetric*])
finally show $g\ e = 0$ **using** *Pair eq*[*THEN fun-cong*, *of e*]
by(*cases f e g e rule*: *ennreal2-cases*)(*simp-all add*: *fun-eq-iff*)
qed
thus $g = (\lambda\text{-}.\ 0)$ **by**(*simp add*: *fun-eq-iff*)
next
have 0: *current* $?\Gamma$ *zero-current* **by** *simp*
show $\neg$ *hindrance* $?\Gamma$ *zero-current*
proof
assume *hindrance* $?\Gamma$ *zero-current*
then obtain x **where** a: $x \in A\ ?\Gamma$ **and** $\mathcal{E}$: $x \notin \mathcal{E}_{?\Gamma}\ (TER_{?\Gamma}\ zero\text{-}current)$

and *d-OUT zero-current* x < *weight* ?Γ x **by** *cases*
from *a* **have** $x \in \mathcal{E}$ (*TER f*) **by** *simp*
then obtain *p y* **where** *p*: *path* Γ *x p y* **and** *y*: $y \in B$ Γ
and *bypass*: $\bigwedge z.$ $[\![x \neq y;\ z \in$ *set p*$]\!] \Longrightarrow z = x \vee z \notin$ *TER f* **by**(*rule* $\mathcal{E}$-*E*) *blast*
from *p* **obtain** p' **where** p': *path* Γ *x* p' *y* **and** *distinct*: *distinct* (x # p')
and *subset*: *set* $p' \subseteq$ *set p* **by**(*auto elim*: *rtrancl-path-distinct*)
note p'
moreover have *RF*: $z \notin$ *RF* (*TER f*) **if** $z \in$ *set* p' **for** *z*
proof
assume *z*: $z \in$ *RF* (*TER f*)
from *split-list*[*OF that*] **obtain** *ys zs* **where** *decomp*: p' = *ys* @ *z* # *zs* **by** *blast*
with p' **have** $y \in$ *set* p' **by**(*auto dest*!: *rtrancl-path-last intro*: *last-in-set*)
with *distinct* **have** *neq*: $x \neq y$ **by** *auto*
from *decomp* p' **have** *path* Γ *z zs y* **by**(*auto elim*: *rtrancl-path-appendE*)
from *roofedD*[*OF z this y*] **obtain** z' **where** $z' \in$ *set* (*z* # *zs*) $z' \in$ *TER f* **by** *auto*
with *distinct decomp subset bypass*[*OF neq*] **show** *False* **by** *auto*
qed
moreover have $x \notin$ RF° (*TER f*) **using** ‹$x \in \mathcal{E}$ (*TER f*)› **by**(*simp add*: *roofed-circ-def*)
ultimately have p'': *path* ?Γ *x* p' *y*
by(*induction*)(*auto intro*: *rtrancl-path.intros simp add*: *roofed-circ-def*)
from *a* $\mathcal{E}$ **have** ¬ *essential* ?Γ (*B* ?Γ) ($TER_{?\Gamma}$ *zero-current*) *x* **by** *simp*
from *not-essentialD*[*OF this* p''] *y* **obtain** *z* **where** *neq*: $x \neq y$
and $z \in$ *set* p' $z \neq x$ $z \in$ $TER_{?\Gamma}$ *zero-current* **by** *auto*
moreover with *subset RF*[*of z*] **have** $z \in$ *TER f*
using *currentD-weight-OUT*[*OF f*, *of z*] *currentD-weight-IN*[*OF f*, *of z*]
by(*auto simp add*: *roofed-circ-def SINK.simps intro*: *SAT.IN split*: *if-split-asm*)
ultimately show *False* **using** *bypass*[*of z*] *subset* **by** *auto*
qed
qed

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*)
proof(*rule web.equality*)
from *trimming* **have** $\mathcal{E}$: $\mathcal{E}$ (*TER g*) − *A* Γ = $\mathcal{E}$ (*TER f*) − *A* Γ **by** *cases*

have *RF*: *RF* (*TER g*) = *RF* (*TER f*)
by(*subst* (*1 2*) *RF-essential*[*symmetric*])(*simp only*: *trimming-*$\mathcal{E}$[*OF w trimming*])
have *RFc*: RF° (*TER g*) = RF° (*TER f*)
by(*subst* (*1 2*) *roofed-circ-essential*[*symmetric*])(*simp only*: *trimming-*$\mathcal{E}$[*OF w trimming*])

show *edge ?lhs* = *edge ?rhs* **by**(*rule ext*)+(*simp add*: *RF RFc*)
have *weight ?lhs* = (λx. *if* $x \in RF^\circ$ (*TER g*) $\vee$ $x \in RF$ (*TER g*) $\cap$ *B* Γ *then 0 else weight* Γ *x*)
unfolding *RF RFc* **by**(*auto simp add*: *fun-eq-iff RF-in-B*)
also have ... = *weight ?rhs* **by**(*auto simp add*: *fun-eq-iff RF-in-B*)
finally show *weight ?lhs* = *weight ?rhs* **.**

show *A ?lhs* = *A ?rhs* **unfolding** *quotient-web-sel trimming-*$\mathcal{E}$[*OF w trimming*]
..
qed *simp-all*

6.8 Well-formed webs

locale *web* =
fixes Γ :: ('*v*, '*more*) *web-scheme* (**structure**)
assumes *A-in*: $x \in A\ \Gamma \Longrightarrow \neg$ *edge* Γ *y x*
and *B-out*: $x \in B\ \Gamma \Longrightarrow \neg$ *edge* Γ *x y*
and *A-vertex*: $A\ \Gamma \subseteq \mathbf{V}$
and *disjoint*: $A\ \Gamma \cap B\ \Gamma = \{\}$
and *no-loop*: $\bigwedge x.\ \neg$ *edge* Γ *x x*
and *weight-outside*: $\bigwedge x.\ x \notin \mathbf{V} \Longrightarrow$ *weight* Γ *x* = *0*
and *weight-finite* [*simp*]: $\bigwedge x.$ *weight* Γ $x \neq \top$
begin

lemma *web-weight-update*:
assumes $\bigwedge x.\ \neg$ *vertex* Γ $x \Longrightarrow w\ x = 0$
and $\bigwedge x.\ w\ x \neq \top$
shows *web* ($\Gamma(\!|$*weight* := *w*$|\!)$)
by *unfold-locales*(*simp-all add*: *A-in B-out A-vertex disjoint no-loop assms*)

lemma *currentI* [*intro?*]:
assumes $\bigwedge x.$ *d-OUT f x* $\leq$ *weight* Γ *x*
and $\bigwedge x.$ *d-IN f x* $\leq$ *weight* Γ *x*
and *OUT-IN*: $\bigwedge x.\ [\![\ x \notin A\ \Gamma;\ x \notin B\ \Gamma\ ]\!] \Longrightarrow$ *d-OUT f x* $\leq$ *d-IN f x*
and *outside*: $\bigwedge e.\ e \notin \mathbf{E} \Longrightarrow f\ e = 0$
shows *current* Γ *f*
proof
show *d-IN f a* = *0* **if** $a \in A\ \Gamma$ **for** *a* **using** *that*
by(*auto simp add*: *d-IN-def nn-integral-0-iff emeasure-count-space-eq-0 A-in intro*: *outside*)
show *d-OUT f b* = *0* **if** $b \in B\ \Gamma$ **for** *b* **using** *that*
by(*auto simp add*: *d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0 B-out intro*: *outside*)
then show *d-OUT f x* $\leq$ *d-IN f x* **if** $x \notin A\ \Gamma$ **for** *x* **using** *OUT-IN*[*OF that*]
by(*cases* $x \in B\ \Gamma$) *auto*
qed(*blast intro*: *assms*)+

lemma *currentD-finite-IN*:
assumes *f*: *current* Γ *f*

shows *d-IN f x* $\neq \top$
proof(*cases* $x \in$ **V**)
case *True*
have *d-IN f x* $\leq$ *weight* Γ *x* **using** *f* **by**(*rule currentD-weight-IN*)
also have ... $< \top$ **using** *True weight-finite*[*of x*] **by** (*simp add*: *less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
next
case *False*
then have *d-IN f x* = *0*
by(*auto simp add*: *d-IN-def nn-integral-0-iff emeasure-count-space-eq-0 vertex-def intro*: *currentD-outside*[*OF f*])
thus *?thesis* **by** *simp*
qed

lemma *currentD-finite-OUT*:
assumes *f*: *current* Γ *f*
shows *d-OUT f x* $\neq \top$
proof(*cases* $x \in$ **V**)
case *True*
have *d-OUT f x* $\leq$ *weight* Γ *x* **using** *f* **by**(*rule currentD-weight-OUT*)
also have ... $< \top$ **using** *True weight-finite*[*of x*] **by** (*simp add*: *less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
next
case *False*
then have *d-OUT f x* = *0*
by(*auto simp add*: *d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0 vertex-def intro*: *currentD-outside*[*OF f*])
thus *?thesis* **by** *simp*
qed

lemma *currentD-finite*:
assumes *f*: *current* Γ *f*
shows *f e* $\neq \top$
proof(*cases e*)
case (*Pair x y*)
have *f* (*x*, *y*) $\leq$ *d-OUT f x* **by** (*rule d-OUT-ge-point*)
also have ... $< \top$ **using** *currentD-finite-OUT*[*OF f*] **by** (*simp add*: *less-top*[*symmetric*])
finally show *?thesis* **by**(*simp add*: *Pair*)
qed

lemma *web-quotient-web*: *web* (*quotient-web* Γ *f*) (**is** *web* ?Γ)
proof
show $\neg$ *edge* ?Γ *y x* **if** $x \in A$?Γ **for** *x y* **using** *that* **by**(*auto intro*: *roofed-greaterI*)
show $\neg$ *edge* ?Γ *x y* **if** $x \in B$?Γ **for** *x y* **using** *that* **by**(*auto simp add*: *B-out*)
show *A* ?Γ $\cap$ *B* ?Γ = {} **using** *disjoint* **by** *auto*
show *A* ?Γ $\subseteq$ $\mathbf{V}_{?\Gamma}$
proof
fix *x*
assume $x \in A$?Γ

```
    hence ℰ: x ∈ ℰ (TER f) and x: x ∉ B Γ using disjoint by auto
    from this(1) obtain p y where p: path Γ x p y and y: y ∈ B Γ and bypass:
⋀z. z ∈ set p ⟹ z ∉ RF (TER f)
      by(rule ℰ-E-RF) blast
    from p y x have p ≠ [] by(auto simp add: rtrancl-path-simps)
    with rtrancl-path-nth[OF p, of 0] have edge Γ x (p ! 0) by simp
    moreover have x ∉ RF° (TER f) using ℰ by(simp add: roofed-circ-def)
    moreover have p ! 0 ∉ RF (TER f) using bypass ‹p ≠ []› by auto
    ultimately have edge ?Γ x (p ! 0) by simp
    thus x ∈ V_?Γ by(auto intro: vertexI1)
  qed
  show ¬ edge ?Γ x x for x by(simp add: no-loop)
  show weight ?Γ x = 0 if x ∉ V_?Γ for x
  proof(cases x ∈ RF° (TER f) ∨ x ∈ TER f ∩ B Γ)
    case True thus ?thesis by simp
  next
    case False
    hence RF: x ∉ RF° (TER f) and B: x ∈ B Γ ⟹ x ∉ TER f by auto
    from RF obtain p y where p: path Γ x p y and y: y ∈ B Γ and bypass: ⋀z.
z ∈ set p ⟹ z ∉ RF (TER f)
      apply(cases x ∉ RF (RF (TER f)))
       apply(auto elim!: not-roofedE)[1]
      apply(auto simp add: roofed-circ-def roofed-idem elim: essentialE-RF)
      done
    from that have p = [] using p y B RF bypass
      by(auto 4 3 simp add: vertex-def dest!: rtrancl-path-nth[where i=0])
    with p have xy: x = y by(simp add: rtrancl-path-simps)
    with B y have x ∉ TER f by simp
    hence RF': x ∉ RF (TER f) using xy y by(subst RF-in-B) auto
    have ¬ vertex Γ x
    proof
      assume vertex Γ x
      then obtain x' where edge Γ x' x using xy y by(auto simp add: vertex-def
B-out)
     moreover hence x' ∉ RF° (TER f) using RF' by(auto dest: RF-circ-edge-forward)
      ultimately have edge ?Γ x' x using RF' by simp
      hence vertex ?Γ x by(rule vertexI2)
      with that show False by simp
    qed
    thus ?thesis by(simp add: weight-outside)
  qed
  show weight ?Γ x ≠ ⊤ for x by simp
qed

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**
by(*simp add*: **V**-*def*)

lemma *countable-web-quotient-web*: *countable-web* (*quotient-web* Γ *f*) (**is** *countable-web ?Γ*)
proof −
interpret *r*: *web ?Γ* **by**(*rule web-quotient-web*)
show *?thesis*
proof
have $\mathbf{E}_{?\Gamma} \subseteq \mathbf{E}$ **by** *auto*
then show *countable* $\mathbf{E}_{?\Gamma}$ **by**(*rule countable-subset*) *simp*
qed
qed

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 Γ ⊖ *f* = Γ⦇*weight* := λ*x*. *if x* ∈ *A* Γ *then weight* Γ *x* − *d-OUT f x else weight* Γ *x* + *d-OUT f x* − *d-IN f x*⦈

lemma *minus-web-sel* [*simp*]:
edge (Γ ⊖ *f*) = *edge* Γ
weight (Γ ⊖ *f*) *x* = (*if x* ∈ *A* Γ *then weight* Γ *x* − *d-OUT f x else weight* Γ *x* + *d-OUT f x* − *d-IN f x*)
A (Γ ⊖ *f*) = *A* Γ
B (Γ ⊖ *f*) = *B* Γ
$\mathbf{V}_{\Gamma \ominus f} = \mathbf{V}_{\Gamma}$
$\mathbf{E}_{\Gamma \ominus f} = \mathbf{E}_{\Gamma}$
web.more (Γ ⊖ *f*) = *web.more* Γ
by(*auto simp add*: *minus-web-def vertex-def*)

lemma *vertex-minus-web* [*simp*]: *vertex* (Γ ⊖ *f*) = *vertex* Γ
by(*simp add*: *vertex-def fun-eq-iff*)

lemma *roofed-gen-minus-web* [*simp*]: *roofed-gen* (Γ ⊖ *f*) = *roofed-gen* Γ
by(*simp add*: *fun-eq-iff roofed-def*)

lemma *minus-zero-current* [*simp*]: Γ ⊖ *zero-current* = Γ
by(*rule web.equality*)(*simp-all add*: *fun-eq-iff*)

lemma (**in** *web*) *web-minus-web*:
assumes *f*: *current* Γ *f*
shows *web* (Γ ⊖ *f*)

unfolding *minus-web-def*
apply(*rule web-weight-update*)
apply(*auto simp*: *weight-outside currentD-weight-IN*[*OF f*] *currentD-outside-OUT*[*OF f*]
currentD-outside-IN[*OF f*] *currentD-weight-OUT*[*OF f*] *currentD-finite-OUT*[*OF f*])
done

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* **E**
begin

lemma *not-vertex*: $[\![\ x \notin A\ \Gamma;\ x \notin B\ \Gamma\ ]\!] \Longrightarrow \neg$ *vertex* Γ *x*
using *bipartite-V* **by** *blast*

lemma *no-loop*: $\neg$ *edge* Γ *x x*
using *disjoint* **by**(*auto dest*: *bipartite-E*)

lemma *edge-antiparallel*: *edge* Γ *x y* $\Longrightarrow \neg$ *edge* Γ *y x*
using *disjoint* **by**(*auto dest*: *bipartite-E*)

lemma *A-in*: $x \in A\ \Gamma \Longrightarrow \neg$ *edge* Γ *y x*
using *disjoint* **by**(*auto dest*: *bipartite-E*)

lemma *B-out*: $x \in B\ \Gamma \Longrightarrow \neg$ *edge* Γ *x y*
using *disjoint* **by**(*auto dest*: *bipartite-E*)

sublocale *countable-web* **using** *disjoint*
by(*unfold-locales*)(*auto simp add*: *A-in B-out A-vertex no-loop weight-outside*)

lemma *currentD-OUT'*:
assumes *f*: *current* Γ *f*
and *x*: $x \notin A\ \Gamma$
shows *d-OUT f x* = *0*
using *currentD-outside-OUT*[*OF f*, *of x*] *x currentD-OUT*[*OF f*, *of x*] *bipartite-V*
by *auto*

lemma *currentD-IN'*:
assumes *f*: *current* Γ *f*
and *x*: $x \notin B\ \Gamma$

shows *d-IN f x = 0*
using *currentD-outside-IN*[*OF f, of x*] *x currentD-IN*[*OF f, of x*] *bipartite-V* **by** *auto*

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* ∉ **E** ⟹ *f e = 0*
shows *current* Γ *f*
proof
fix *x*
assume *x* ∉ *A* Γ *x* ∉ *B* Γ
hence *d-OUT f x = 0* **by**(*auto simp add*: *d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0 intro*!: *outside dest*: *bipartite-E*)
then show *d-OUT f x* ≤ *d-IN f x* **by** *simp*
qed(*rule OUT IN outside*)+

lemma *wave-bipartiteI* [*intro?*]:
assumes *sep*: *separating* Γ (*TER f*)
and *f*: *current* Γ *f*
shows *wave* Γ *f*
using *sep*
proof(*rule wave.intros*)
fix *x*
assume *x* ∉ *RF* (*TER f*)
then consider *x* ∉ **V** | *x* ∈ **V** *x* ∈ *B* Γ **using** *separating-RF-A*[*OF sep*] *bipartite-V* **by** *auto*
then show *d-OUT f x = 0* **using** *currentD-OUT*[*OF f, of x*] *currentD-outside-OUT*[*OF f, of x*]
by *cases auto*
qed

lemma *web-flow-iff*: *web-flow* Γ *f* ⟷ *current* Γ *f*
using *bipartite-V* **by**(*auto simp add*: *web-flow.simps*)

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* = $\lambda uv\ uv'$. *case* $(uv,\ uv')$ *of* (*Inl u*, *Inr v*) $\Rightarrow$ *edge* $\Gamma\ u\ v \vee u = v \wedge u \in$ *vertices* $\Gamma \wedge u \notin A\ \Gamma \wedge v \notin B\ \Gamma$ | - $\Rightarrow$ *False*,
   weight = λuv. *case uv of Inl u* $\Rightarrow$ *if* $u \in B\ \Gamma$ *then 0 else weight* $\Gamma\ u$ | *Inr u* $\Rightarrow$ *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* ' ($- A\ \Gamma$)
by(*simp-all add*: *bipartite-web-of-def split*: *sum.split*)

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

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*)
by(*auto*)

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* $u \in \mathbf{V}\ u \notin A\ \Gamma\ u \notin B\ \Gamma$
using *assms* **by**(*cases uv uv' rule*: *sum.exhaust*[*case-product sum.exhaust*]) *auto*

lemma *E-bipartite-web*:
  fixes Γ (**structure**) **shows**
  $\mathbf{E}_{bipartite\text{-}web\text{-}of\ \Gamma} = (\lambda(x,\ y).\ (Inl\ x,\ Inr\ y))$ ' $\mathbf{E} \cup (\lambda x.\ (Inl\ x,\ Inr\ x))$ ' $(\mathbf{V} - A\ \Gamma - B\ \Gamma)$
by(*auto elim*: *edge-bipartite-webE*)

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* $\Gamma\ x \wedge x \notin A\ \Gamma$
by(*auto 4 4 simp add*: *vertex-def dest*: *B-out A-in intro*: *edge-bipartite-webI1 edge-bipartite-webI2 elim*!: *edge-bipartite-webE*)

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* Γ ∩ *A-S*) ∪ (*B* Γ ∩ *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* ∈ *B* Γ
and *x*: *x* ∈ *A* Γ ∨ *Inr x* ∈ *S*
shows (∃ *z*∈*set p*. *z* ∈ *separating-of-bipartite S*) ∨ *x* ∈ *separating-of-bipartite S*
proof(*cases p* = [])
case *True*
with *p* **have** *x* = *y* **by** *cases auto*
with *y x* **have** *x* ∈ *separating-of-bipartite S* **using** *disjoint*
by(*auto simp add*: *separating-of-bipartite-def Let-def*)
thus *?thesis* **..**
next
case *nNil*: *False*
define *inmarked* **where** *inmarked x* ⟷ *x* ∈ *A* Γ ∨ *Inr x* ∈ *S* **for** *x*
define *outmarked* **where** *outmarked x* ⟷ *x* ∈ *B* Γ ∨ *Inl x* ∈ *S* **for** *x*
let *?*Γ = *bipartite-web-of* Γ
let *?double* = λ*x*. *inmarked x* ∧ *outmarked x*
define *tailmarked* **where** *tailmarked* = (λ(*x*, *y* :: $'v$). *outmarked x*)
define *headmarked* **where** *headmarked* = (λ(*x* :: $'v$, *y*). *inmarked y*)

have *marked-E*: *tailmarked e* ∨ *headmarked e* **if** *e* ∈ **E** **for** *e* — Lemma 1b
proof(*cases e*)
case (*Pair x y*)
with *that* **have** *path ?*Γ (*Inl x*) [*Inr y*] (*Inr y*) **by**(*auto intro*!: *rtrancl-path.intros*)
from *separatingD*[*OF sep this*] *that Pair* **show** *?thesis*
by(*fastforce simp add*: *vertex-def inmarked-def outmarked-def tailmarked-def headmarked-def*)
qed

have ∃ *z*∈*set* (*x* # *p*). *?double z* — Lemma 2
proof −
have *inmarked* ((*x* # *p*) ! (*i* + *1*)) ∨ *outmarked* ((*x* # *p*) ! *i*) **if** *i* < *length p* **for** *i*
using *rtrancl-path-nth*[*OF p that*] *marked-E*[*of* ((*x* # *p*) ! *i*, *p* ! *i*)] *that*
by(*auto simp add*: *tailmarked-def headmarked-def nth-Cons split*: *nat.split*)
hence {*i*. *i* < *length p* ∧ *inmarked* (*p* ! *i*)} ∪ {*i*. *i* < *length* (*x* # *butlast p*) ∧ *outmarked* ((*x* # *butlast p*) ! *i*)} = {*i*. *i* < *length p*}

(**is** *?in* ∪ *?out* = -) **using** *nNil*
by(*force simp add*: *nth-Cons′ nth-butlast elim*: *meta-allE*[**where** *x*=*0*] *cong del*: *old.nat.case-cong-weak*)
hence *length p* + *2* = *card* (*?in* ∪ *?out*) + *2* **by** *simp*
also have ... ≤ (*card ?in* + *1*) + (*card ?out* + *1*) **by**(*simp add*: *card-Un-le*)
also have *card ?in* = *card* ((λ*i*. *Inl* (*i* + *1*) :: - + *nat*) ‘ *?in*)
by(*rule card-image*[*symmetric*])(*simp add*: *inj-on-def*)
also have ... + *1* = *card* (*insert* (*Inl 0*) {*Inl* (*Suc i*) :: - + *nat*|*i*. *i* < *length p* ∧ *inmarked* (*p* ! *i*)})
by(*subst card-insert-if*)(*auto intro*!: *arg-cong*[**where** *f*=*card*])
also have ... = *card* {*Inl i* :: *nat* + *nat*|*i*. *i* < *length* (*x* # *p*) ∧ *inmarked* ((*x* # *p*) ! *i*)} (**is** - = *card ?in*)
using *x* **by**(*intro arg-cong*[**where** *f*=*card*])(*auto simp add*: *nth-Cons inmarked-def split*: *nat.split-asm*)
also have *card ?out* = *card* ((*Inr* :: - ⇒ *nat* + -) ‘ *?out*) **by**(*simp add*: *card-image*)
also have ... + *1* = *card* (*insert* (*Inr* (*length p*)) {*Inr i* :: *nat* + -|*i*. *i* < *length p* ∧ *outmarked* ((*x* # *p*) ! *i*)})
using *nNil* **by**(*subst card-insert-if*)(*auto intro*!: *arg-cong*[**where** *f*=*card*] *simp add*: *nth-Cons nth-butlast cong*: *nat.case-cong*)
also have ... = *card* {*Inr i* :: *nat* + -|*i*. *i* < *length* (*x* # *p*) ∧ *outmarked* ((*x* # *p*) ! *i*)} (**is** - = *card ?out*)
using *nNil rtrancl-path-last*[*OF p nNil*] *y*
by(*intro arg-cong*[**where** *f*=*card*])(*auto simp add*: *outmarked-def last-conv-nth*)
also have *card ?in* + *card ?out* = *card* (*?in* ∪ *?out*)
by(*rule card-Un-disjoint*[*symmetric*]) *auto*
also let *?f* = *case-sum id id*
have *?f* ‘ (*?in* ∪ *?out*) ⊆ {*i*. *i* < *length* (*x* # *p*)} **by** *auto*
from *card-mono*[*OF* - *this*] **have** *card* (*?f* ‘ (*?in* ∪ *?out*)) ≤ *length p* + *1* **by** *simp*
ultimately have ¬ *inj-on ?f* (*?in* ∪ *?out*) **by**(*intro pigeonhole*) *simp*
then obtain *i* **where** *i* < *length* (*x* # *p*) *?double* ((*x* # *p*) ! *i*)
by(*auto simp add*: *inj-on-def*)
thus *?thesis* **by**(*auto simp add*: *set-conv-nth*)
qed
moreover have *z* ∈ *separating-of-bipartite S* **if** *?double z* **for** *z* **using** *that disjoint*
by(*auto simp add*: *separating-of-bipartite-def Let-def inmarked-def outmarked-def*)
ultimately show *?thesis* **by** *auto*
qed

lemma *separating-of-bipartite*:
separating Γ (*separating-of-bipartite S*)
by(*rule separating-gen.intros*)(*erule* (*1*) *separating-of-bipartite-aux*; *simp*)

end

lemma *current-bipartite-web-finite*:
assumes *f*: *current* (*bipartite-web-of* Γ) *f* (**is** *current ?*Γ -)
shows *f e* ≠ ⊤

proof(*cases e*)
  case (*Pair x y*)
  have *f e* ≤ *d-OUT f x* **unfolding** *Pair d-OUT-def* **by**(*rule nn-integral-ge-point*) *simp*
  also have ... ≤ *weight ?Γ x* **by**(*rule currentD-weight-OUT*[*OF f*])
  also have ... < ⊤ **by**(*cases x*)(*simp-all add*: *less-top*[*symmetric*])
  finally show *?thesis* **by** *simp*
qed

definition *current-of-bipartite* :: (′*v* + ′*v*) *current* ⇒ ′*v current*
where *current-of-bipartite f* = (λ(*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*)
by(*simp add*: *current-of-bipartite-def*)

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 −
  have *d-OUT* (*current-of-bipartite f*) *x* = ∫ + *y*. *f* (*Inl x*, *y*) ∗ *indicator* **E** (*x*, *projr y*) ∂*count-space* (*range Inr*)
    by(*simp add*: *d-OUT-def nn-integral-count-space-reindex*)
  also have ... = *d-OUT f* (*Inl x*) − ∫ + *y*. *f* (*Inl x*, *y*) ∗ *indicator* {*Inr x*} *y* ∂*count-space UNIV* (**is** - = - − *?rest*)
    unfolding *d-OUT-def* **by**(*subst nn-integral-diff*[*symmetric*])(*auto 4 4 simp add*: *current-bipartite-web-finite*[*OF f*] *AE-count-space nn-integral-count-space-indicator no-loop split*: *split-indicator intro*!: *nn-integral-cong intro*: *currentD-outside*[*OF f*] *elim*: *edge-bipartite-webE*)
  finally show *?thesis* **by** *simp*
qed

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 −
  have *d-IN* (*current-of-bipartite f*) *x* = ∫ + *y*. *f* (*y*, *Inr x*) ∗ *indicator* **E** (*projl y*, *x*) ∂*count-space* (*range Inl*)
    by(*simp add*: *d-IN-def nn-integral-count-space-reindex*)
  also have ... = *d-IN f* (*Inr x*) − ∫ + *y*. *f* (*y*, *Inr x*) ∗ *indicator* {*Inl x*} *y* ∂*count-space UNIV* (**is** - = - − *?rest*)
    unfolding *d-IN-def* **by**(*subst nn-integral-diff*[*symmetric*])(*auto 4 4 simp add*: *current-bipartite-web-finite*[*OF f*] *AE-count-space nn-integral-count-space-indicator no-loop split*: *split-indicator intro*!: *nn-integral-cong intro*: *currentD-outside*[*OF f*] *elim*: *edge-bipartite-webE*)
  finally show *?thesis* **by** *simp*
qed

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
fix *x*
have *d-OUT ?f x* ≤ *d-OUT f* (*Inl x*)
by(*simp add*: *d-OUT-current-of-bipartite*[*OF f*] *diff-le-self-ennreal*)
also have ... ≤ *weight* Γ *x*
using *currentD-weight-OUT*[*OF f*, *of Inl x*]
by(*simp split*: *if-split-asm*)
finally show *d-OUT ?f x* ≤ *weight* Γ *x* **.**
next
fix *x*
have *d-IN ?f x* ≤ *d-IN f* (*Inr x*)
by(*simp add*: *d-IN-current-of-bipartite*[*OF f*] *diff-le-self-ennreal*)
also have ... ≤ *weight* Γ *x*
using *currentD-weight-IN*[*OF f*, *of Inr x*]
by(*simp split*: *if-split-asm*)
finally show *d-IN ?f x* ≤ *weight* Γ *x* **.**
next
have *OUT*: *d-OUT ?f b* = *0* **if** *b* ∈ *B* Γ **for** *b* **using** *that*
by(*auto simp add*: *d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0 intro*!: *currentD-outside*[*OF f*] *dest*: *B-out*)
show *d-OUT ?f x* ≤ *d-IN ?f x* **if** *A*: *x* ∉ *A* Γ **for** *x*
proof(*cases x* ∈ *B* Γ ∨ *x* ∉ **V**)
case *True*
then show *?thesis*
proof
assume *x* ∈ *B* Γ
with *OUT*[*OF this*] **show** *?thesis* **by** *auto*
next
assume *x* ∉ **V**
hence *d-OUT ?f x* = *0* **by**(*auto simp add*: *d-OUT-def vertex-def nn-integral-0-iff emeasure-count-space-eq-0 intro*!: *currentD-outside*[*OF f*])
thus *?thesis* **by** *simp*
qed
next
case *B* [*simplified*]: *False*
have *d-OUT ?f x* = *d-OUT f* (*Inl x*) − *f* (*Inl x*, *Inr x*) (**is** - = - − *?rest*)
by(*simp add*: *d-OUT-current-of-bipartite*[*OF f*])
also have *d-OUT f* (*Inl x*) ≤ *d-IN f* (*Inr x*)
proof(*rule ccontr*)
assume ¬ *?thesis*
hence ∗: *d-IN f* (*Inr x*) < *d-OUT f* (*Inl x*) **by**(*simp add*: *not-less*)
also have ... ≤ *weight* Γ *x* **using** *currentD-weight-OUT*[*OF f*, *of Inl x*] *B* **by** *simp*
finally have *Inr x* ∉ $TER_{?\Gamma}$ *f* **using** *A* **by**(*auto elim*!: *SAT.cases*)
moreover have *Inl x* ∉ $TER_{?\Gamma}$ *f* **using** ∗ **by**(*auto simp add*: *SINK.simps*)
moreover have *path ?*Γ (*Inl x*) [*Inr x*] (*Inr x*)

by(*rule rtrancl-path.step*)(*auto intro*!: *rtrancl-path.base simp add*: *no-loop A B*)
ultimately show *False* **using** *waveD-separating*[*OF w*] *A B* **by**(*auto dest*!: *separatingD*)
qed
hence *d-OUT f* (*Inl x*) $-$ *?rest* $\leq$ *d-IN f* (*Inr x*) $-$ *?rest* **by**(*rule ennreal-minus-mono*) *simp*
also have ... = *d-IN ?f x* **by**(*simp add*: *d-IN-current-of-bipartite*[*OF f*])
finally show *?thesis* **.**
qed
show *?f e* = *0* **if** $e \notin$ **E** **for** *e* **using** *that* **by**(*cases e*)(*auto*)
qed

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(*rule set-eqI*)
fix *x*
consider (*A*) $x \in A\ \Gamma\ x \in$ **V** | (*B*) $x \in B\ \Gamma\ x \in$ **V**
| (*inner*) $x \notin A\ \Gamma\ x \notin B\ \Gamma\ x \in$ **V** | (*outside*) $x \notin$ **V** **by** *auto*
thus $x \in$ *TER ?f* $\longleftrightarrow$ $x \in$ *separating-of-bipartite ?TER*
proof *cases*
case *A*
hence *d-OUT ?f x* = *d-OUT f* (*Inl x*) **using** *currentD-outside*[*OF f*, *of Inl x Inr x*]
by(*simp add*: *d-OUT-current-of-bipartite*[*OF f*] *no-loop*)
thus *?thesis* **using** *A disjoint*
by(*auto simp add*: *separating-of-bipartite-def Let-def SINK.simps intro*!: *SAT.A imageI*)
next
case *B*
then have *d-IN ?f x* = *d-IN f* (*Inr x*)
using *currentD-outside*[*OF f*, *of Inl x Inr x*]
by(*simp add*: *d-IN-current-of-bipartite*[*OF f*] *no-loop*)
moreover have *d-OUT ?f x* = *0* **using** *B currentD-outside*[*OF f*, *of Inl x Inr x*]
by(*simp add*: *d-OUT-current-of-bipartite*[*OF f*] *no-loop*)(*auto simp add*: *d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0 intro*!: *currentD-outside*[*OF f*] *elim*!: *edge-bipartite-webE dest*: *B-out*)
moreover have *d-OUT f* (*Inr x*) = *0* **using** *B disjoint* **by**(*intro currentD-OUT*[*OF f*]) *auto*
ultimately show *?thesis* **using** *B*
by(*auto simp add*: *separating-of-bipartite-def Let-def SINK.simps SAT.simps*)
next
case *outside*
with *current-current-of-bipartite*[*OF f w*] **have** *d-OUT ?f x* = *0* *d-IN ?f x* = *0*

by(*rule currentD-outside-OUT currentD-outside-IN*)+
moreover from *outside* **have** *Inl x* ∉ *vertices* ?Γ *Inr x* ∉ *vertices* ?Γ **by** *auto*
hence *d-OUT f* (*Inl x*) = *0* *d-IN f* (*Inl x*) = *0* *d-OUT f* (*Inr x*) = *0* *d-IN f* (*Inr x*) = *0*
by(*blast intro*: *currentD-outside-OUT*[*OF f*] *currentD-outside-IN*[*OF f*])+
ultimately show *?thesis* **using** *outside weight-outside*[*of x*]
by(*auto simp add*: *separating-of-bipartite-def Let-def SINK.simps SAT.simps not-le*)
next
case *inner*
show *?thesis*
proof
assume *x* ∈ *separating-of-bipartite ?TER*
with *inner* **have** *l*: *Inl x* ∈ *?TER* **and** *r*: *Inr x* ∈ *?TER*
by(*auto simp add*: *separating-of-bipartite-def Let-def*)
have *f* (*Inl x*, *Inr x*) ≤ *d-OUT f* (*Inl x*)
unfolding *d-OUT-def* **by**(*rule nn-integral-ge-point*) *simp*
with *l* **have** *0*: *f* (*Inl x*, *Inr x*) = *0*
by(*auto simp add*: *SINK.simps*)
with *l* **have** *x* ∈ *SINK ?f* **by**(*simp add*: *SINK.simps d-OUT-current-of-bipartite*[*OF f*])
moreover from *r* **have** *Inr x* ∈ *SAT* ?Γ *f* **by** *auto*
with *inner* **have** *x* ∈ *SAT* Γ *?f*
by(*auto elim*!: *SAT.cases intro*!: *SAT.IN simp add*: *0 d-IN-current-of-bipartite*[*OF f*])
ultimately show *x* ∈ *TER ?f* **by** *simp*
next
assume ∗: *x* ∈ *TER ?f*
have *d-IN f* (*Inr x*) ≤ *weight* ?Γ (*Inr x*) **using** *f* **by**(*rule currentD-weight-IN*)
also have . . . ≤ *weight* Γ *x* **using** *inner* **by** *simp*
also have . . . ≤ *d-IN ?f x* **using** *inner* ∗ **by**(*auto elim*: *SAT.cases*)
also have . . . ≤ *d-IN f* (*Inr x*)
by(*simp add*: *d-IN-current-of-bipartite*[*OF f*] *max-def diff-le-self-ennreal*)
ultimately have *eq*: *d-IN ?f x* = *d-IN f* (*Inr x*) **by** *simp*
hence *0*: *f* (*Inl x*, *Inr x*) = *0*
using *ennreal-minus-cancel-iff*[*of d-IN f* (*Inr x*) *f* (*Inl x*, *Inr x*) *0*] *currentD-weight-IN*[*OF f*, *of Inr x*] *inner*
d-IN-ge-point[*of f Inl x Inr x*]
by(*auto simp add*: *d-IN-current-of-bipartite*[*OF f*] *top-unique*)
have *Inl x* ∈ *?TER Inr x* ∈ *?TER* **using** *inner* ∗ *currentD-OUT*[*OF f*, *of Inr x*]
by(*auto simp add*: *SAT.simps SINK.simps d-OUT-current-of-bipartite*[*OF f*] *0 eq*)
thus *x* ∈ *separating-of-bipartite ?TER* **unfolding** *separating-of-bipartite-def Let-def* **by** *blast*
qed
qed
qed

```
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
  have sep′: separating Γ (separating-of-bipartite (TER?Γ f))
    by(rule separating-of-bipartite)(rule waveD-separating[OF w])
  then show sep: separating Γ (TER (current-of-bipartite f))
    by(simp add: TER-current-of-bipartite[OF f w])

  fix x
  assume x ∉ RF (TER ?f)
  then obtain p y where p: path Γ x p y and y: y ∈ B Γ and x: x ∉ TER ?f
    and bypass: ⋀z. z ∈ set p ⟹ z ∉ TER ?f
    by(auto simp add: roofed-def elim: rtrancl-path-distinct)
  from p obtain p′ where p′: path Γ x p′ y and distinct: distinct (x # p′)
    and subset: set p′ ⊆ set p by(auto elim: rtrancl-path-distinct)
  consider (outside) x ∉ V | (A) x ∈ A Γ | (B) x ∈ B Γ | (inner) x ∈ V x ∉ A
Γ x ∉ B Γ by auto
  then show d-OUT ?f x = 0
  proof cases
    case outside
   with f w show ?thesis by(rule currentD-outside-OUT[OF current-current-of-bipartite])
  next
    case A
    from separatingD[OF sep p A y] bypass have x ∈ TER ?f by blast
    thus ?thesis by(simp add: SINK.simps)
  next
    case B
    with f w show ?thesis by(rule currentD-OUT[OF current-current-of-bipartite])
  next
    case inner
    hence path ?Γ (Inl x) [Inr x] (Inr x) by(auto intro!: rtrancl-path.intros)
    from inner waveD-separating[OF w, THEN separatingD, OF this]
    consider (Inl) Inl x ∈ TER?Γ f | (Inr) Inr x ∈ TER?Γ f by auto
    then show ?thesis
    proof cases
      case Inl
      thus ?thesis
        by(auto simp add: SINK.simps d-OUT-current-of-bipartite[OF f] max-def)
    next
      case Inr
      with separating-of-bipartite-aux[OF waveD-separating[OF w] p y] x bypass
      have False unfolding TER-current-of-bipartite[OF f w] by blast
      thus ?thesis ..
    qed
  qed
qed
```

end

context *countable-web* **begin**

lemma *countable-bipartite-web-of*: *countable-bipartite-web* (*bipartite-web-of* Γ) (**is** *countable-bipartite-web ?Γ*)
proof
  show $\mathbf{V}_{?\Gamma} \subseteq A\ ?\Gamma \cup B\ ?\Gamma$
    apply(*rule subsetI*)
    subgoal for *x* **by**(*cases x*) *auto*
    done
  show $A\ ?\Gamma \subseteq \mathbf{V}_{?\Gamma}$ **by** *auto*
  show $x \in A\ ?\Gamma \wedge y \in B\ ?\Gamma$ **if** *edge ?Γ x y* **for** *x y* **using** *that*
    by(*cases x y rule*: *sum.exhaust*[*case-product sum.exhaust*])(*auto simp add*: *inj-image-mem-iff vertex-def B-out A-in*)
  show $A\ ?\Gamma \cap B\ ?\Gamma = \{\}$ **by** *auto*
  show *countable* $\mathbf{E}_{?\Gamma}$ **by**(*simp add*: *E-bipartite-web*)
  show *weight ?Γ x* $\neq \top$ **for** *x* **by**(*cases x*) *simp-all*
  show *weight* (*bipartite-web-of* Γ) $x = 0$ **if** $x \notin \mathbf{V}_{?\Gamma}$ **for** *x* **using** *that*
    by(*cases x*)(*auto simp add*: *weight-outside*)
qed

end

context *web* **begin**

lemma *unhindered-bipartite-web-of*:
  assumes *loose*: *loose* Γ
  shows ¬ *hindered* (*bipartite-web-of* Γ)
proof
  assume *hindered* (*bipartite-web-of* Γ) (**is** *hindered ?Γ*)
  then obtain *f* **where** *f*: *current ?Γ f* **and** *w*: *wave ?Γ f* **and** *hind*: *hindrance ?Γ f* **by** *cases*
  from *f w* **have** *current* Γ (*current-of-bipartite f*) **by**(*rule current-current-of-bipartite*)
  moreover from *f w* **have** *wave* Γ (*current-of-bipartite f*) **by**(*rule wave-current-of-bipartite*)
  ultimately have ∗: *current-of-bipartite f* = *zero-current* **by**(*rule looseD-wave*[*OF loose*])
  have *zero*: $f\ (Inl\ x,\ Inr\ y) = 0$ **if** $x \neq y$ **for** *x y* **using** *that* ∗[*THEN fun-cong*, *of* (*x*, *y*)]
    by(*cases edge* Γ *x y*)(*auto intro*: *currentD-outside*[*OF f*])

  have *OUT*: *d-OUT f* (*Inl x*) = *f* (*Inl x*, *Inr x*) **for** *x*
  proof −
    have *d-OUT f* (*Inl x*) = ($\sum^+$ *y*. *f* (*Inl x*, *y*) ∗ *indicator* {*Inr x*} *y*) **unfolding** *d-OUT-def*
      using *zero currentD-outside*[*OF f*]
      apply(*intro nn-integral-cong*)
      subgoal for *y* **by**(*cases y*)(*auto split*: *split-indicator*)
      done

also have ... $=$ *f* (*Inl x*, *Inr x*) **by** *simp*
finally show *?thesis* **.**
qed
have *IN*: *d-IN f* (*Inr x*) $=$ *f* (*Inl x*, *Inr x*) **for** *x*
proof $-$
have *d-IN f* (*Inr x*) $=$ ($\sum^+$ *y*. *f* (*y*, *Inr x*) $*$ *indicator* {*Inl x*} *y*) **unfolding** *d-IN-def*
using *zero currentD-outside*[*OF f*]
apply(*intro nn-integral-cong*)
subgoal for *y* **by**(*cases y*)(*auto split*: *split-indicator*)
done
also have ... $=$ *f* (*Inl x*, *Inr x*) **by** *simp*
finally show *?thesis* **.**
qed

let *?TER* $=$ *TER*$_{?\Gamma}$ *f*
from *hind* **obtain** *a* **where** *a*: $a \in A$ *?*Γ **and** $n\mathcal{E}$: $a \notin \mathcal{E}_{?\Gamma}$ (*TER*$_{?\Gamma}$ *f*)
and *OUT-a*: *d-OUT f a* $<$ *weight ?*Γ *a* **by** *cases*
from *a* **obtain** a' **where** a': $a =$ *Inl* a' **and** *v*: *vertex* Γ a' **and** *b*: $a' \notin B\ \Gamma$ **by** *auto*
have *A*: $a' \in A\ \Gamma$
proof(*rule ccontr*)
assume *A*: $a' \notin A\ \Gamma$
hence *edge ?*Γ (*Inl* a') (*Inr* a') **using** *v b* **by** *simp*
hence *p*: *path ?*Γ (*Inl* a') [*Inr* a'] (*Inr* a') **by**(*simp add*: *rtrancl-path-simps*)
from *separatingD*[*OF waveD-separating*[*OF w*] *this*] *b v A*
have *Inl* $a' \in$ *?TER* $\vee$ *Inr* $a' \in$ *?TER* **by** *auto*
thus *False*
proof *cases*
case *left*
hence *d-OUT f* (*Inl* a') $=$ *0* **by**(*simp add*: *SINK.simps*)
moreover hence *d-IN f* (*Inr* a') $=$ *0* **by**(*simp add*: *OUT IN*)
ultimately have *Inr* $a' \notin$ *?TER* **using** *v b A OUT-a* a' **by**(*auto simp add*: *SAT.simps*)
then have *essential ?*Γ (*B ?*Γ) *?TER* (*Inl* a') **using** *A*
by(*intro essentialI*[*OF p*]) *simp-all*
with $n\mathcal{E}$ *left* a' **show** *False* **by** *simp*
next
case *right*
hence *d-IN f* (*Inr* a') $=$ *weight* Γ a' **using** *A* **by**(*auto simp add*: *currentD-SAT*[*OF f*])
hence *d-OUT f* (*Inl* a') $=$ *weight* Γ a' **by**(*simp add*: *IN OUT*)
with *OUT-a* a' *b* **show** *False* **by** *simp*
qed
qed
moreover

from *A* **have** *d-OUT f* (*Inl* a') $=$ *0* **using** *currentD-outside*[*OF f*, *of Inl* a' *Inr* a']

```
    by(simp add: OUT no-loop)
  with b v have TER: Inl a′ ∈ ?TER by(simp add: SAT.A SINK.simps)
  with nℰ a′ have ness: ¬ essential ?Γ (B ?Γ) ?TER (Inl a′) by simp

  have a′ ∉ ℰ (TER zero-current)
  proof
    assume a′ ∈ ℰ (TER zero-current)
    then obtain p y where p: path Γ a′ p y and y: y ∈ B Γ
      and bypass: ⋀z. z ∈ set p ⟹ z ∉ TER zero-current
      by(rule ℰ-E-RF)(auto intro: roofed-greaterI)

    from p show False
    proof cases
      case base with y A disjoint show False by auto
    next
      case (step x p′)
     from step(2) have path ?Γ (Inl a′) [Inr x] (Inr x) by(simp add: rtrancl-path-simps)
      from not-essentialD[OF ness this] bypass[of x] step(1)
      have Inr x ∈ ?TER by simp
      with bypass[of x] step(1) have d-IN f (Inr x) > 0
        by(auto simp add: currentD-SAT[OF f] zero-less-iff-neq-zero)
      hence x: Inl x ∉ ?TER by(auto simp add: SINK.simps OUT IN)
      from step(1) have set (x # p′) ⊆ set p by auto
      with ‹path Γ x p′ y› x y show False
      proof induction
        case (base x)
      thus False using currentD-outside-IN[OF f, of Inl x] currentD-outside-OUT[OF
f, of Inl x]
          by(auto simp add: currentD-SAT[OF f] SINK.simps dest!: bypass)
      next
        case (step x z p′ y)
          from step.prems(3) bypass[of x] weight-outside[of x] have x: vertex Γ x
by(auto)
           from ‹edge Γ x z› have path ?Γ (Inl x) [Inr z] (Inr z) by(simp add:
rtrancl-path-simps)
      from separatingD[OF waveD-separating[OF w] this] step.prems(1) step.prems(3)
bypass[of z] x ‹edge Γ x z›
        have Inr z ∈ ?TER by(force simp add: B-out inj-image-mem-iff)
        with bypass[of z] step.prems(3) ‹edge Γ x z› have d-IN f (Inr z) > 0
          by(auto simp add: currentD-SAT[OF f] A-in zero-less-iff-neq-zero)
        hence x: Inl z ∉ ?TER by(auto simp add: SINK.simps OUT IN)
        with step.IH[OF this] step.prems(2,3) show False by auto
      qed
    qed
  qed
  moreover have d-OUT zero-current a′ < weight Γ a′
    using OUT-a a′ b by (auto simp: zero-less-iff-neq-zero)
  ultimately have hindrance Γ zero-current by(rule hindrance)
  with looseD-hindrance[OF loose] show False by contradiction
```

qed

lemma (**in** −) *divide-less-1-iff-ennreal*: a / b < (*1*::*ennreal*) ⟷ (*0* < b ∧ a < b ∨ b = *0* ∧ a = *0* ∨ b = *top*)
  by (*cases a*; *cases b*; *cases b* = *0*)
    (*auto simp*: *divide-ennreal ennreal-less-iff ennreal-top-divide*)

lemma *linkable-bipartite-web-ofD*:
  assumes *link*: *linkable* (*bipartite-web-of* Γ) (**is** *linkable ?*Γ)
  and *countable*: *countable* **E**
  shows *linkable* Γ
proof −
  from *link* **obtain** *f* **where** *wf*: *web-flow ?*Γ *f* **and** *link*: *linkage ?*Γ *f* **by** *blast*
  from *wf* **have** *f*: *current ?*Γ *f* **by**(*rule web-flowD-current*)
  define *f′* **where** *f′* = *current-of-bipartite f*

  have *IN-le-OUT*: *d-IN f′ x* ≤ *d-OUT f′ x* **if** *x* ∉ *B* Γ **for** *x*
  proof(*cases x* ∈ **V**)
    case *True*
    have *d-IN f′ x* = *d-IN f* (*Inr x*) − *f* (*Inl x*, *Inr x*) (**is** - = - − *?rest*)
      by(*simp add*: *f′-def d-IN-current-of-bipartite*[*OF f*])
    also have ... ≤ *weight ?*Γ (*Inr x*) − *?rest*
      using *currentD-weight-IN*[*OF f*, *of Inr x*] **by**(*rule ennreal-minus-mono*) *simp*
     also have ... ≤ *weight ?*Γ (*Inl x*) − *?rest* **using** *that ennreal-minus-mono* **by**(*auto*)
    also have *weight ?*Γ (*Inl x*) = *d-OUT f* (*Inl x*) **using** *that linkageD*[*OF link*, *of Inl x*] *True* **by** *auto*
    also have ... − *?rest* = *d-OUT f′ x*
      by(*simp add*: *f′-def d-OUT-current-of-bipartite*[*OF f*])
    finally show *?thesis* **.**
  next
    case *False*
    with *currentD-outside-OUT*[*OF f*, *of Inl x*] *currentD-outside-IN*[*OF f*, *of Inr x*]
    show *?thesis* **by**(*simp add*: *f′-def d-IN-current-of-bipartite*[*OF f*] *d-OUT-current-of-bipartite*[*OF f*])
  qed
  have *link*: *linkage* Γ *f′*
  proof
    show *d-OUT f′ a* = *weight* Γ *a* **if** *a* ∈ *A* Γ **for** *a*
    proof(*cases a* ∈ **V**)
      case *True*
      from *that* **have** *a* ∉ *B* Γ **using** *disjoint* **by** *auto*
      with *that True linkageD*[*OF link*, *of Inl a*] *ennreal-minus-cancel-iff*[*of - - 0*] *currentD-outside*[*OF f*, *of Inl a Inr a*]
      show *?thesis* **by**(*clarsimp simp add*: *f′-def d-OUT-current-of-bipartite*[*OF f*] *max-def no-loop*)
    next
      case *False*

with *weight-outside*[*OF this*] *currentD-outside*[*OF f*, *of Inl a Inr a*] *currentD-outside-OUT*[*OF f*, *of Inl a*]
show *?thesis* **by**(*simp add*: *f'-def d-OUT-current-of-bipartite*[*OF f*] *no-loop*)
qed
qed

define *F* **where** $F = \{g.\ (\forall e.\ 0 \leq g\ e) \wedge (\forall e.\ e \notin \mathbf{E} \longrightarrow g\ e = 0) \wedge$
$(\forall x.\ x \notin B\ \Gamma \longrightarrow \textit{d-IN}\ g\ x \leq \textit{d-OUT}\ g\ x) \wedge$
$\textit{linkage}\ \Gamma\ g \wedge$
$(\forall x{\in}A\ \Gamma.\ \textit{d-IN}\ g\ x = 0) \wedge$
$(\forall x.\ \textit{d-OUT}\ g\ x \leq \textit{weight}\ \Gamma\ x) \wedge$
$(\forall x.\ \textit{d-IN}\ g\ x \leq \textit{weight}\ \Gamma\ x) \wedge$
$(\forall x{\in}B\ \Gamma.\ \textit{d-OUT}\ g\ x = 0) \wedge g \leq f'\}$
define *leq* **where** $\textit{leq} = \textit{restrict-rel}\ F\ \{(f, f').\ f' \leq f\}$
have *F*: $\textit{Field leq} = F$ **by**(*auto simp add*: *leq-def*)
have *F-I* [*intro?*]: $f \in \textit{Field leq}$ **if** $\bigwedge e.\ 0 \leq f\ e$ **and** $\bigwedge e.\ e \notin \mathbf{E} \Longrightarrow f\ e = 0$
and $\bigwedge x.\ x \notin B\ \Gamma \Longrightarrow \textit{d-IN}\ f\ x \leq \textit{d-OUT}\ f\ x$ **and** $\textit{linkage}\ \Gamma\ f$
and $\bigwedge x.\ x \in A\ \Gamma \Longrightarrow \textit{d-IN}\ f\ x = 0$ **and** $\bigwedge x.\ \textit{d-OUT}\ f\ x \leq \textit{weight}\ \Gamma\ x$
and $\bigwedge x.\ \textit{d-IN}\ f\ x \leq \textit{weight}\ \Gamma\ x$ **and** $\bigwedge x.\ x \in B\ \Gamma \Longrightarrow \textit{d-OUT}\ f\ x = 0$
and $f \leq f'$ **for** *f* **using** *that* **by**(*simp add*: *F F-def*)
have *F-nonneg*: $0 \leq f\ e$ **if** $f \in \textit{Field leq}$ **for** *f e* **using** *that* **by**(*cases e*)(*simp add*: *F F-def*)
have *F-outside*: $f\ e = 0$ **if** $f \in \textit{Field leq}\ e \notin \mathbf{E}$ **for** *f e* **using** *that* **by**(*cases e*)(*simp add*: *F F-def*)
have *F-IN-OUT*: $\textit{d-IN}\ f\ x \leq \textit{d-OUT}\ f\ x$ **if** $f \in \textit{Field leq}\ x \notin B\ \Gamma$ **for** *f x* **using** *that* **by**(*simp add*: *F F-def*)
have *F-link*: $\textit{linkage}\ \Gamma\ f$ **if** $f \in \textit{Field leq}$ **for** *f* **using** *that* **by**(*simp add*: *F F-def*)
have *F-IN*: $\textit{d-IN}\ f\ x = 0$ **if** $f \in \textit{Field leq}\ x \in A\ \Gamma$ **for** *f x* **using** *that* **by**(*simp add*: *F F-def*)
have *F-OUT*: $\textit{d-OUT}\ f\ x = 0$ **if** $f \in \textit{Field leq}\ x \in B\ \Gamma$ **for** *f x* **using** *that* **by**(*simp add*: *F F-def*)
have *F-weight-OUT*: $\textit{d-OUT}\ f\ x \leq \textit{weight}\ \Gamma\ x$ **if** $f \in \textit{Field leq}$ **for** *f x* **using** *that* **by**(*simp add*: *F F-def*)
have *F-weight-IN*: $\textit{d-IN}\ f\ x \leq \textit{weight}\ \Gamma\ x$ **if** $f \in \textit{Field leq}$ **for** *f x* **using** *that* **by**(*simp add*: *F F-def*)
have *F-le*: $f\ e \leq f'\ e$ **if** $f \in \textit{Field leq}$ **for** *f e* **using** *that* **by**(*cases e*)(*simp add*: *F F-def le-fun-def*)

have *F-finite-OUT*: $\textit{d-OUT}\ f\ x \neq \top$ **if** $f \in \textit{Field leq}$ **for** *f x*
proof −
have $\textit{d-OUT}\ f\ x \leq \textit{weight}\ \Gamma\ x$ **by**(*rule F-weight-OUT*[*OF that*])
also have $\ldots < \top$ **by**(*simp add*: *less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

have *F-finite*: $f\ e \neq \top$ **if** $f \in \textit{Field leq}$ **for** *f e*
proof(*cases e*)
case (*Pair x y*)
have $f\ e \leq \textit{d-OUT}\ f\ x$ **unfolding** *Pair d-OUT-def* **by**(*rule nn-integral-ge-point*)

simp
also have ... $<$ $\top$ **by**(*simp add*: *F-finite-OUT*[*OF that*] *less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

have f': $f' \in$ *Field leq*
proof
show $0 \leq f'\ e$ **for** e **by**(*cases e*)(*simp add*: *f'-def*)
show $f'\ e = 0$ **if** $e \notin$ **E** **for** e **using** *that* **by**(*clarsimp split*: *split-indicator-asm simp add*: *f'-def*)
show *d-IN* $f'\ x \leq$ *d-OUT* $f'\ x$ **if** $x \notin B\ \Gamma$ **for** x **using** *that* **by**(*rule IN-le-OUT*)
show *linkage* $\Gamma\ f'$ **by**(*rule link*)
show *d-IN* $f'\ x = 0$ **if** $x \in A\ \Gamma$ **for** x **using** *that currentD-IN*[*OF f*, *of Inl x*] *disjoint*
currentD-outside[*OF f*, *of Inl x Inr x*] *currentD-outside-IN*[*OF f*, *of Inr x*]
by(*cases* $x \in$ **V**)(*auto simp add*: *d-IN-current-of-bipartite*[*OF f*] *no-loop f'-def*)
show *d-OUT* $f'\ x = 0$ **if** $x \in B\ \Gamma$ **for** x **using** *that currentD-OUT*[*OF f*, *of Inr x*] *disjoint*
currentD-outside[*OF f*, *of Inl x Inr x*] *currentD-outside-OUT*[*OF f*, *of Inl x*]
by(*cases* $x \in$ **V**)(*auto simp add*: *d-OUT-current-of-bipartite*[*OF f*] *no-loop f'-def*)
show *d-OUT* $f'\ x \leq$ *weight* $\Gamma\ x$ **for** x **using** *currentD-weight-OUT*[*OF f*, *of Inl x*]
by(*simp add*: *d-OUT-current-of-bipartite*[*OF f*] *ennreal-diff-le-mono-left f'-def split*: *if-split-asm*)
show *d-IN* $f'\ x \leq$ *weight* $\Gamma\ x$ **for** x **using** *currentD-weight-IN*[*OF f*, *of Inr x*]
by(*simp add*: *d-IN-current-of-bipartite*[*OF f*] *ennreal-diff-le-mono-left f'-def split*: *if-split-asm*)
qed *simp*

have *F-leI*: $g \in$ *Field leq* **if** f: $f \in$ *Field leq* **and** *le*: $\bigwedge e.\ g\ e \leq f\ e$
and *nonneg*: $\bigwedge e.\ 0 \leq g\ e$ **and** *IN-OUT*: $\bigwedge x.\ x \notin B\ \Gamma \Longrightarrow$ *d-IN* $g\ x \leq$ *d-OUT* $g\ x$
and *link*: *linkage* $\Gamma\ g$
for $f\ g$
proof
show $g\ e = 0$ **if** $e \notin$ **E** **for** e **using** *nonneg*[*of e*] *F-outside*[*OF f that*] *le*[*of e*] **by** *simp*
show *d-IN* $g\ a = 0$ **if** $a \in A\ \Gamma$ **for** a
using *d-IN-mono*[*of g a f*, *OF le*] *F-IN*[*OF f that*] **by** *auto*
show *d-OUT* $g\ b = 0$ **if** $b \in B\ \Gamma$ **for** b
using *d-OUT-mono*[*of g b f*, *OF le*] *F-OUT*[*OF f that*] **by** *auto*
show *d-OUT* $g\ x \leq$ *weight* $\Gamma\ x$ **for** x
using *d-OUT-mono*[*of g x f*, *OF le*] *F-weight-OUT*[*OF f*] **by**(*rule order-trans*)
show *d-IN* $g\ x \leq$ *weight* $\Gamma\ x$ **for** x
using *d-IN-mono*[*of g x f*, *OF le*] *F-weight-IN*[*OF f*] **by**(*rule order-trans*)
show $g \leq f'$ **using** *order-trans*[*OF le F-le*[*OF f*]] **by**(*auto simp add*: *le-fun-def*)
qed(*blast intro*: *IN-OUT link nonneg*)+

have *chain-Field*: *Inf M* $\in$ *F* **if** *M*: *M* $\in$ *Chains leq* **and** *nempty*: *M* $\neq$ {} **for** *M*
proof −
from *nempty* **obtain** *g0* **where** *g0-in-M*: *g0* $\in$ *M* **by** *auto*
with *M* **have** *g0*: *g0* $\in$ *Field leq* **by**(*rule Chains-FieldD*)

from *M* **have** *M* $\in$ *Chains* $\{(g, g'). \ g' \leq g\}$
by(*rule mono-Chains*[*THEN subsetD*, *rotated*])(*auto simp add*: *leq-def in-restrict-rel-iff*)
then have *Complete-Partial-Order.chain* ($\geq$) *M* **by**(*rule Chains-into-chain*)
hence *chain'*: *Complete-Partial-Order.chain* ($\leq$) *M* **by**(*simp add*: *chain-dual*)

have *support-flow f'* $\subseteq$ **E** **using** *F-outside*[*OF f'*] **by**(*auto intro*: *ccontr simp add*: *support-flow.simps*)
then have *countable'*: *countable* (*support-flow f'*)
by(*rule countable-subset*)(*simp add*: *E-bipartite-web countable* **V**-*def*)

have *finite-OUT*: *d-OUT f' x* $\neq$ $\top$ **for** *x* **using** *weight-finite*[*of x*]
by(*rule neq-top-trans*)(*rule F-weight-OUT*[*OF f'*])
have *finite-IN*: *d-IN f' x* $\neq$ $\top$ **for** *x* **using** *weight-finite*[*of x*]
by(*rule neq-top-trans*)(*rule F-weight-IN*[*OF f'*])
have *OUT-M*: *d-OUT* (*Inf M*) *x* = (*INF g*$\in$*M*. *d-OUT g x*) **for** *x* **using** *chain' nempty countable' - finite-OUT*
by(*rule d-OUT-Inf*)(*auto dest*!: *Chains-FieldD*[*OF M*] *simp add*: *leq-def F-nonneg F-le*)
have *IN-M*: *d-IN* (*Inf M*) *x* = (*INF g*$\in$*M*. *d-IN g x*) **for** *x* **using** *chain' nempty countable' - finite-IN*
by(*rule d-IN-Inf*)(*auto dest*!: *Chains-FieldD*[*OF M*] *simp add*: *leq-def F-nonneg F-le*)

show *Inf M* $\in$ *F* **using** *g0* **unfolding** *F*[*symmetric*]
proof(*rule F-leI*)
show (*Inf M*) *e* $\leq$ *g0 e* **for** *e* **using** *g0-in-M* **by**(*auto intro*: *INF-lower*)
show *0* $\leq$ (*Inf M*) *e* **for** *e* **by**(*auto intro*!: *INF-greatest dest*: *F-nonneg dest*!: *Chains-FieldD*[*OF M*])
show *d-IN* (*Inf M*) *x* $\leq$ *d-OUT* (*Inf M*) *x* **if** *x* $\notin$ *B* Γ **for** *x* **using** *that*
by(*auto simp add*: *IN-M OUT-M intro*!: *INF-mono dest*: *Chains-FieldD*[*OF M*] *intro*: *F-IN-OUT*[*OF - that*])
show *linkage* Γ (*Inf M*) **using** *nempty*
by(*simp add*: *linkage.simps OUT-M F-link*[*THEN linkageD*] *Chains-FieldD*[*OF M*] *cong*: *INF-cong*)
qed
qed

let *?P* = $\lambda g\ z.\ z \notin A\ \Gamma \wedge z \notin B\ \Gamma \wedge$ *d-OUT g z* > *d-IN g z*

define *link*
where *link g* =
(*if* $\exists$ *z*. *?P g z then*
let z = *SOME z*. *?P g z*; *factor* = *d-IN g z* / *d-OUT g z*
in ($\lambda(x, y)$. (*if x* = *z then factor else 1*) $*$ *g* (*x*, *y*))

```
      else g) for g
  have increasing: link g ≤ g ∧ link g ∈ Field leq if g: g ∈ Field leq for g
  proof(cases ∃ z. ?P g z)
    case False
    thus ?thesis using that by(auto simp add: link-def leq-def)
  next
    case True
    define z where z = Eps (?P g)
    from True have ?P g z unfolding z-def by(rule someI-ex)
    hence A: z ∉ A Γ and B: z ∉ B Γ and less: d-IN g z < d-OUT g z by simp-all
    let ?factor = d-IN g z / d-OUT g z
    have link [simp]: link g (x, y) = (if x = z then ?factor else 1) * g (x, y) for x y
      using True by(auto simp add: link-def z-def Let-def)

    have ?factor ≤ 1 (is ?factor ≤ -) using less
     by(cases d-OUT g z d-IN g z rule: ennreal2-cases) (simp-all add: ennreal-less-iff
divide-ennreal)
    hence le': ?factor * g (x, y) ≤ 1 * g (x, y) for y x
      by(rule mult-right-mono)(simp add: F-nonneg[OF g])
    hence le: link g e ≤ g e for e by(cases e)simp
    have link g ∈ Field leq using g le
    proof(rule F-leI)
      show nonneg: 0 ≤ link g e for e
        using F-nonneg[OF g, of e] by(cases e) simp
       show linkage Γ (link g) using that A F-link[OF g] by(clarsimp simp add:
linkage.simps d-OUT-def)

      fix x
      assume x: x ∉ B Γ
     have d-IN (link g) x ≤ d-IN g x unfolding d-IN-def using le' by(auto intro:
nn-integral-mono)
      also have ... ≤ d-OUT (link g) x
      proof(cases x = z)
        case True
        have d-IN g x = d-OUT (link g) x unfolding d-OUT-def
            using True F-weight-IN[OF g, of x] F-IN-OUT[OF g x] F-finite-OUT
F-finite-OUT[OF g, of x]
          by(cases d-OUT g z = 0)
         (auto simp add: nn-integral-divide nn-integral-cmult d-OUT-def[symmetric]
ennreal-divide-times less-top)
        thus ?thesis by simp
      next
        case False
        have d-IN g x ≤ d-OUT g x using x by(rule F-IN-OUT[OF g])
          also have ... ≤ d-OUT (link g) x unfolding d-OUT-def using False
by(auto intro!: nn-integral-mono)
        finally show ?thesis .
      qed
      finally show d-IN (link g) x ≤ d-OUT (link g) x .
```

qed
with *le g* **show** *?thesis* **unfolding** *F* **by**(*simp add*: *leq-def le-fun-def del*: *link*)
qed

have *bourbaki-witt-fixpoint Inf leq link* **using** *chain-Field increasing* **unfolding** *leq-def*
by(*intro bourbaki-witt-fixpoint-restrict-rel*)(*auto intro*: *Inf-greatest Inf-lower*)
then interpret *bourbaki-witt-fixpoint Inf leq link* **.**

define *g* **where** $g = \mathit{fixp\text{-}above}\ f'$

have *g*: $g \in \mathit{Field\ leq}$ **using** f' **unfolding** *g-def* **by**(*rule fixp-above-Field*)
hence *linkage* Γ *g* **by**(*rule F-link*)
moreover
have *web-flow* Γ *g*
proof(*intro web-flow.intros current.intros*)
show $\mathit{d\text{-}OUT}\ g\ x \leq \mathit{weight}\ \Gamma\ x$ **for** *x* **using** *g* **by**(*rule F-weight-OUT*)
show $\mathit{d\text{-}IN}\ g\ x \leq \mathit{weight}\ \Gamma\ x$ **for** *x* **using** *g* **by**(*rule F-weight-IN*)
show $\mathit{d\text{-}IN}\ g\ x = 0$ **if** $x \in A\ \Gamma$ **for** *x* **using** *g that* **by**(*rule F-IN*)
show *B*: $\mathit{d\text{-}OUT}\ g\ x = 0$ **if** $x \in B\ \Gamma$ **for** *x* **using** *g that* **by**(*rule F-OUT*)
show $g\ e = 0$ **if** $e \notin \mathbf{E}$ **for** *e* **using** *g that* **by**(*rule F-outside*)

show *KIR*: *KIR g x* **if** *A*: $x \notin A\ \Gamma$ **and** *B*: $x \notin B\ \Gamma$ **for** *x*
proof(*rule ccontr*)
define *z* **where** $z = \mathit{Eps}\ (?P\ g)$
assume $\neg$ *KIR g x*
with *F-IN-OUT*[*OF g B*] **have** $\mathit{d\text{-}OUT}\ g\ x > \mathit{d\text{-}IN}\ g\ x$ **by** *simp*
with *A B* **have** *Ex*: $\exists x.\ ?P\ g\ x$ **by** *blast*
then have *?P g z* **unfolding** *z-def* **by**(*rule someI-ex*)
hence *A*: $z \notin A\ \Gamma$ **and** *B*: $z \notin B\ \Gamma$ **and** *less*: $\mathit{d\text{-}IN}\ g\ z < \mathit{d\text{-}OUT}\ g\ z$ **by** *simp-all*
let $?factor = \mathit{d\text{-}IN}\ g\ z\ /\ \mathit{d\text{-}OUT}\ g\ z$
have $\exists y.\ \mathit{edge}\ \Gamma\ z\ y \wedge g\ (z,\ y) > 0$
proof(*rule ccontr*)
assume $\neg$ *?thesis*
hence $\mathit{d\text{-}OUT}\ g\ z = 0$ **using** *F-outside*[*OF g*]
by(*force simp add*: *d-OUT-def nn-integral-0-iff-AE AE-count-space not-less*)
with *less* **show** *False* **by** *simp*
qed
then obtain *y* **where** *y*: *edge* Γ *z y* **and** *gr0*: $g\ (z,\ y) > 0$ **by** *blast*
have $?factor < 1$ (**is** $?factor < -$) **using** *less*
by(*cases d-OUT g z d-IN g z rule*: *ennreal2-cases*)
(*auto simp add*: *ennreal-less-iff divide-ennreal*)

hence *le'*: $?factor * g\ (z,\ y) < 1 * g\ (z,\ y)$ **using** *gr0 F-finite*[*OF g*]
by(*intro ennreal-mult-strict-right-mono*) (*auto simp*: *less-top*)
hence $\mathit{link}\ g\ (z,\ y) \neq g\ (z,\ y)$ **using** *Ex* **by**(*auto simp add*: *link-def z-def Let-def*)
hence $\mathit{link}\ g \neq g$ **by**(*auto simp add*: *fun-eq-iff*)

moreover have *link* $g = g$ **using** f' **unfolding** *g-def* **by**(*rule fixp-above-unfold*[*symmetric*])
ultimately show *False* **by** *contradiction*
qed
show *d-OUT* $g\ x \leq$ *d-IN* $g\ x$ **if** $x \notin A\ \Gamma$ **for** x **using** *KIR*[*of x*] *that B*[*of x*]
by(*cases* $x \in B\ \Gamma$) *auto*
qed
ultimately show *?thesis* **by** *blast*
qed

end

7.2 Extending a wave by a linkage

lemma *linkage-quotient-webD*:
fixes Γ :: (*'v*, *'more*) *web-scheme* (**structure**) **and** *h g*
defines $k \equiv$ *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 −
from *wg* **have** *g*: *current ?*Γ *g* **by**(*rule web-flowD-current*)

from *trim* **obtain** *h*: *current* Γ *h* **and** *w'*: *wave* Γ *h* **and** *h-le-f*: $\bigwedge e.\ h\ e \leq f\ e$
and *KIR*: $\bigwedge x.\ [\![x \in RF^{\circ}\ (TER\ f);\ x \notin A\ \Gamma]\!] \Longrightarrow KIR\ h\ x$
and *TER*: *TER h* $\supseteq \mathcal{E}$ (*TER f*) − *A* Γ
by(*cases*)(*auto simp add*: *le-fun-def*)

have *eq*: *quotient-web* Γ *f* = *quotient-web* Γ *h* **using** *w trim* **by**(*rule quotient-web-trimming*)

let *?T* = $\mathcal{E}$ (*TER f*)

have *RFc*: RF° (*TER h*) = RF° (*TER f*)
by(*subst* (*1 2*) *roofed-circ-essential*[*symmetric*])(*simp only*: *trimming-*$\mathcal{E}$[*OF w trim*])
have *OUT-k*: *d-OUT k x* = (*if* $x \in RF^{\circ}$ (*TER f*) *then d-OUT h x else d-OUT g x*) **for** *x*
using *OUT-plus-current*[*OF h w'*, *of g*] *web-flowD-current*[*OF wg*] **unfolding** *eq k-def RFc* **by** *simp*
have *RF*: *RF* (*TER h*) = *RF* (*TER f*)
by(*subst* (*1 2*) *RF-essential*[*symmetric*])(*simp only*: *trimming-*$\mathcal{E}$[*OF w trim*])
have *IN-k*: *d-IN k x* = (*if* $x \in RF$ (*TER f*) *then d-IN h x else d-IN g x*) **for** *x*
using *IN-plus-current*[*OF h w'*, *of g*] *web-flowD-current*[*OF wg*] **unfolding** *eq k-def RF* **by** *simp*

have *k*: *current* Γ *k* **unfolding** *k-def* **using** *h w′ g* **unfolding** *eq* **by**(*rule current-plus-current*)
then show *wk*: *web-flow* Γ *k*
proof(*rule web-flow*)
fix *x*
assume $x \in$ **V** **and** *A*: $x \notin A\ \Gamma$ **and** *B*: $x \notin B\ \Gamma$
show *KIR k x*
proof(*cases* $x \in \mathcal{E}$ (*TER f*))
case *False*
thus *?thesis* **using** *A KIR*[*of x*] *web-flowD-KIR*[*OF wg*, *of x*] *B* **by**(*auto simp add*: *OUT-k IN-k roofed-circ-def*)
next
case *True*
with *A TER* **have** [*simp*]: *d-OUT h x* = *0* **and** *d-IN h x* ≥ *weight* Γ *x*
by(*auto simp add*: *SINK.simps elim*: *SAT.cases*)
with *currentD-weight-IN*[*OF h*, *of x*] **have** [*simp*]: *d-IN h x* = *weight* Γ *x* **by** *auto*
from *linkageD*[*OF link*, *of x*] *True currentD-IN*[*OF g*, *of x*] *B*
have *d-OUT g x* = *weight* Γ *x d-IN g x* = *0* **by**(*auto simp add*: *roofed-circ-def*)
thus *?thesis* **using** *True* **by**(*auto simp add*: *IN-k OUT-k roofed-circ-def intro*: *roofed-greaterI*)
qed
qed

have *h-le-k*: *h e* ≤ *k e* **for** *e* **unfolding** *k-def plus-current-def* **by**(*rule add-increasing2*) *simp-all*
hence *SAT* Γ *h* ⊆ *SAT* Γ *k* **by**(*rule SAT-mono*)
hence *SAT*: *?T* ⊆ *SAT* Γ *k* **using** *TER* **by**(*auto simp add*: *elim*!: *SAT.cases intro*: *SAT.intros*)
show *orthogonal-current* Γ *k ?T*
proof(*rule orthogonal-current*)
show *weight* Γ *x* ≤ *d-IN k x* **if** *x* ∈ *?T x* ∉ *A* Γ **for** *x*
using *subsetD*[*OF SAT*, *of x*] *that* **by**(*auto simp add*: *currentD-SAT*[*OF k*])
next
fix *x*
assume *x*: *x* ∈ *?T* **and** *A*: *x* ∈ *A* Γ **and** *B*: *x* ∉ *B* Γ
with *d-OUT-mono*[*of h x f*, *OF h-le-f*] **have** *d-OUT h x* = *0* **by**(*auto simp add*: *SINK.simps*)
moreover from *linkageD*[*OF link*, *of x*] *x A* **have** *d-OUT g x* = *weight ?*Γ *x* **by** *simp*
ultimately show *d-OUT k x* = *weight* Γ *x* **using** *x A currentD-IN*[*OF f A*] *B*
by(*auto simp add*: *d-OUT-add roofed-circ-def k-def plus-current-def*)
next
fix *u v*
assume *v*: *v* ∈ *RF ?T* **and** *u*: *u* ∉ *RF*° *?T*
have *h* (*u*, *v*) ≤ *f* (*u*, *v*) **by**(*rule h-le-f*)
also have . . . ≤ *d-OUT f u* **unfolding** *d-OUT-def* **by**(*rule nn-integral-ge-point*) *simp*
also have . . . = *0* **using** *u* **using** *RF-essential*[*of* Γ *TER f*]

by(*auto simp add*: *roofed-circ-def SINK.simps intro*: *waveD-OUT*[*OF w*])
finally have $h\ (u,\ v) = 0$ **by** *simp*
moreover have $g\ (u,\ v) = 0$ **using** *g v RF-essential*[*of* Γ *TER f*]
by(*auto intro*: *currentD-outside simp add*: *roofed-circ-def*)
ultimately show $k\ (u,\ v) = 0$ **by**(*simp add*: *k-def*)
qed
qed

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*) ⟧
⟹ *linkable* (*quotient-web* Γ *f*)
shows $\exists f\ S.$ *web-flow* Γ *f* ∧ *separating* Γ *S* ∧ *orthogonal-current* Γ *f S*
proof −
from *ex-maximal-wave*[*OF countable*]
obtain *f* **where** *f*: *current* Γ *f*
and *w*: *wave* Γ *f*
and *maximal*: $\bigwedge w.$ ⟦ *current* Γ *w*; *wave* Γ *w*; $f \leq w$ ⟧ ⟹ $f = w$ **by** *blast*
from *ex-trimming*[*OF f w countable weight-finite*] **obtain** *h* **where** *h*: *trimming*
Γ *f h* **..**

let *?*Γ = *quotient-web* Γ *f*
interpret Γ: *countable-web ?*Γ **by**(*rule countable-web-quotient-web*)
have *loose ?*Γ **using** *f w maximal* **by**(*rule loose-quotient-web*[*OF weight-finite*])
with *f w* **have** *linkable ?*Γ **by**(*rule loose-linkable*)
then obtain *g* **where** *wg*: *web-flow ?*Γ *g* **and** *link*: *linkage ?*Γ *g* **by** *blast*

let *?k* = *plus-current h g*
have *web-flow* Γ *?k orthogonal-current* Γ *?k* ($\mathcal{E}$ (*TER f*))
by(*rule linkage-quotient-webD*[*OF f w wg link h*])+
moreover have *separating* Γ ($\mathcal{E}$ (*TER f*))
using *waveD-separating*[*OF w*] **by**(*rule separating-essential*)
ultimately show *?thesis* **by** *blast*
qed

end

7.3 From a network to a web

definition *web-of-network* :: (*'v*, *'more*) *network-scheme* ⇒ (*'v edge*, *'more*) *web-scheme*
where
web-of-network Δ =
⦇*edge* = λ(*x*, *y*) (*y'*, *z*). *y'* = *y* ∧ *edge* Δ *x y* ∧ *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* $e' \longleftrightarrow e \in \mathbf{E} \wedge e' \in \mathbf{E} \wedge$ *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)$\}$
by(*auto simp add*: *web-of-network-def split*: *prod.split*)

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*)
by(*auto simp add*: *vertex-def Domainp.simps Rangep.simps*)

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*)
proof
from *fw* **have** *f*: *current* Γ *f* **and** *KIR*: $\bigwedge x.\ [\![\ x \notin A\ \Gamma;\ x \notin B\ \Gamma\ ]\!] \Longrightarrow$ *KIR f x*
by(*auto 4 3 dest*: *web-flowD-current web-flowD-KIR*)

show *?f e* $\leq$ *capacity* Δ *e* **for** *e*
using *currentD-weight-OUT*[*OF f*, *of e*] *currentD-weight-IN*[*OF f*, *of e*]
by(*simp add*: *flow-of-current-def*)

fix *x*
assume *x*: $x \neq$ *source* Δ $x \neq$ *sink* Δ
have *d-OUT ?f x* = ($\sum^+$ *y*. *d-IN f* (*x*, *y*)) **unfolding** *d-OUT-def*
by(*simp add*: *flow-of-current-def max-absorb2 currentD-OUT-IN*[*OF f*] *x*)
also have ... = ($\sum^+$ *y*. $\sum^+$ *e*$\in$*range* (λ*z*. (*z*, *x*)). *f* (*e*, *x*, *y*))
by(*auto simp add*: *nn-integral-count-space-indicator d-IN-def intro*!: *nn-integral-cong currentD-outside*[*OF f*] *split*: *split-indicator*)
also have ... = ($\sum^+$ *z*$\in$*UNIV*. $\sum^+$ *y*. *f* ((*z*, *x*), *x*, *y*))
by(*subst nn-integral-snd-count-space*[*of case-prod* -, *simplified*])
(*simp add*: *nn-integral-count-space-reindex nn-integral-fst-count-space*[*of case-prod* -, *simplified*])
also have ... = ($\sum^+$ *z*. $\sum^+$ *e*$\in$*range* (*Pair x*). *f* ((*z*, *x*), *e*))
by(*simp add*: *nn-integral-count-space-reindex*)
also have ... = ($\sum^+$ *z*. *d-OUT f* (*z*, *x*))
by(*auto intro*!: *nn-integral-cong currentD-outside*[*OF f*] *simp add*: *d-OUT-def nn-integral-count-space-indicator split*: *split-indicator*)
also have ... = ($\sum^+$ *z*$\in$$\{$*z*. *edge* Δ *z x*$\}$. *d-OUT f* (*z*, *x*))
by(*auto intro*!: *nn-integral-cong currentD-outside-OUT*[*OF f*] *simp add*: *nn-integral-count-space-indicator split*: *split-indicator*)

also have ... $= (\sum^+ z \in \{z.\ edge\ \Delta\ z\ x\}.\ max\ (d\text{-}OUT\ f\ (z,\ x))\ (d\text{-}IN\ f\ (z,\ x)))$
proof(*rule nn-integral-cong*)
show $d\text{-}OUT\ f\ (z,\ x) = max\ (d\text{-}OUT\ f\ (z,\ x))\ (d\text{-}IN\ f\ (z,\ x))$
if $z \in space\ (count\text{-}space\ \{z.\ edge\ \Delta\ z\ x\})$ **for** *z* **using** *currentD-IN*[*OF f*] *that*
by(*cases* $z = source\ \Delta$)(*simp-all add*: *max-absorb1 currentD-IN*[*OF f*] *KIR x*)
qed
also have ... $= (\sum^+ z.\ max\ (d\text{-}OUT\ f\ (z,\ x))\ (d\text{-}IN\ f\ (z,\ x)))$
by(*auto intro*!: *nn-integral-cong currentD-outside-OUT*[*OF f*] *currentD-outside-IN*[*OF f*] *simp add*: *nn-integral-count-space-indicator max-def split*: *split-indicator*)
also have ... $= d\text{-}IN\ ?f\ x$ **by**(*simp add*: *flow-of-current-def d-IN-def*)
finally show *KIR ?f x* **.**
qed

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* $\Delta\ V$
and *sink*: $sink\ \Delta \notin V$
shows *cut* $\Delta\ (RF^N\ V)$
proof
show $source\ \Delta \in RF^N\ V$ **by**(*rule roofedI*)(*auto dest*: *separatingD*[*OF sep*])
show $sink\ \Delta \notin RF^N\ V$ **using** *sink* **by**(*auto dest*: *roofedD*[*OF - rtrancl-path.base*])
qed

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* $\Gamma\ S$
and *source-sink*: $source\ \Delta \neq sink\ \Delta$
shows *separating-network* $\Delta\ (fst\ `\ \mathcal{E}\ S)$
proof
define *s t* **where** $s = source\ \Delta$ **and** $t = sink\ \Delta$
fix *p*

assume *p*: *path* Δ *s p t*
with *p source-sink* **have** $p \neq []$ **by** *cases*(*auto simp add*: *s-def t-def*)
with *p* **have** *p'*: *path* Γ (*s*, *hd p*) (*zip p* (*tl p*)) (*last* (*s* # *butlast p*), *t*)
proof(*induction*)
case (*step x y p z*)
then show *?case* **by**(*cases p*)(*auto elim*: *rtrancl-path.cases intro*: *rtrancl-path.intros simp add*: Γ-*def*)
qed *simp*
from *sep* **have** *separating* Γ ($\mathcal{E}$ *S*) **by**(*rule separating-essential*)
from *this p'* **have** $(\exists z \in set\ (zip\ p\ (tl\ p)).\ z \in \mathcal{E}\ S) \lor (s,\ hd\ p) \in \mathcal{E}\ S$
apply(*rule separatingD*)
using *rtrancl-path-nth*[*OF p*, *of 0*] *rtrancl-path-nth*[*OF p*, *of length p* − *1*] ‹*p* ≠ []› *rtrancl-path-last*[*OF p*]
apply(*auto simp add*: Γ-*def s-def t-def hd-conv-nth last-conv-nth nth-butlast nth-Cons' cong*: *if-cong split*: *if-split-asm*)
apply(*metis One-nat-def Suc-leI cancel-comm-monoid-add-class.diff-cancel le-antisym length-butlast length-greater-0-conv list.size*(*3*))+
done
then show $(\exists z \in set\ p.\ z \in fst\ `\ \mathcal{E}\ S) \lor s \in fst\ `\ \mathcal{E}\ S$
by(*auto dest*!: *set-zip-leftD intro*: *rev-image-eqI*)
qed

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* Δ ≠ *sink* Δ
and *sink-out*: $\bigwedge x.\ \neg\ edge\ \Delta\ (sink\ \Delta)\ x$
shows *cut* Δ (*cut-of-sep S*)
unfolding *cut-of-sep-def*
proof(*rule cut-RF-separating*)
show *separating-network* Δ (*fst* ‘ $\mathcal{E}$ *S*) **using** *sep neq* **by**(*rule separating-network-cut-of-sep*)

show *sink* Δ ∉ *fst* ‘ $\mathcal{E}$ *S*
proof
assume *sink* Δ ∈ *fst* ‘ $\mathcal{E}$ *S*
then obtain *x* **where** (*sink* Δ, *x*) ∈ $\mathcal{E}$ *S* **by** *auto*
hence (*sink* Δ, *x*) ∈ **V** **by**(*auto simp add*: Γ-*def dest*!: *essential-vertex*)
then show *False* **by**(*simp add*: Γ-*def sink-out*)
qed
qed

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*: $\bigwedge e.\ e \in \mathbf{E}_{\Delta} \Longrightarrow capacity\ \Delta\ e > 0$

begin

private lemma *f*: *current* Γ *f* **using** *wf* **by**(*rule web-flowD-current*)

lemma *orthogonal-leave-RF*:
  assumes *e*: *edge* Δ *x y*
  and *x*: *x* ∈ (*cut-of-sep S*)
  and *y*: *y* ∉ (*cut-of-sep S*)
  shows (*x*, *y*) ∈ *S*
proof −
  from *y* **obtain** *p* **where** *p*: *path* Δ *y p* (*sink* Δ) **and** *y'*: *y* ∉ *fst* ' ℰ *S*
    and *bypass*: ⋀*z*. *z* ∈ *set p* ⟹ *z* ∉ *fst* ' ℰ *S* **by**(*auto simp add*: *roofed-def cut-of-sep-def* Γ-*def*[*symmetric*])
  from *e p* **have** *p'*: *path* Δ *x* (*y* # *p*) (*sink* Δ) **..**
  from *roofedD*[*OF x*[*unfolded cut-of-sep-def*] *this*] *y' bypass* **have** *x* ∈ *fst* ' ℰ *S* **by**(*auto simp add*: Γ-*def*[*symmetric*])
  then obtain *z* **where** *xz*: (*x*, *z*) ∈ ℰ *S* **by** *auto*
  then obtain *q b* **where** *q*: *path* Γ (*x*, *z*) *q b* **and** *b*: *b* ∈ *B* Γ
    and *distinct*: *distinct* ((*x*, *z*) # *q*) **and** *bypass'*: ⋀*z*. *z* ∈ *set q* ⟹ *z* ∉ *RF S*
    by(*rule* ℰ-*E-RF*) *blast*

  define *p'* **where** *p'* = *y* # *p*
  hence *p'* ≠ [] **by** *simp*
  with *p'* **have** *path* Γ (*x*, *hd p'*) (*zip p'* (*tl p'*)) (*last* (*x* # *butlast p'*), *sink* Δ)
    unfolding *p'*-*def*[*symmetric*]
  proof(*induction*)
    case (*step x y p z*)
    then show *?case*
      by(*cases p*)(*auto elim*: *rtrancl-path.cases intro*: *rtrancl-path.intros simp add*: Γ-*def*)
  qed *simp*
  then have *p''*: *path* Γ (*x*, *y*) (*zip* (*y* # *p*) *p*) (*last* (*x* # *butlast* (*y* # *p*)), *sink* Δ) (**is** *path* - *?y ?p ?t*)
    by(*simp add*: *p'*-*def*)
  have (*?y* # *?p*) ! *length p* = *?t* **using** *rtrancl-path-last*[*OF p'*] *p rtrancl-path-last*[*OF p*]
    apply(*auto split*: *if-split-asm simp add*: *nth-Cons butlast-conv-take take-Suc-conv-app-nth split*: *nat.split elim*: *rtrancl-path.cases*)
    apply(*simp add*: *last-conv-nth*)
    done
  moreover have *length p* < *length* (*?y* # *?p*) **by** *simp*
  ultimately have *t*: *?t* ∈ *B* Γ **using** *rtrancl-path-nth*[*OF p''*, *of length p* − *1*] *e*
    by(*cases p*)(*simp-all add*: Γ-*def split*: *if-split-asm*)

  show *S*: (*x*, *y*) ∈ *S*
  proof(*cases x* = *source* Δ)
    case *True*
    from *separatingD*[*OF separating-essential*, *OF sep p''* - *t*] *e True*
    consider (*z*) *z z'* **where** (*z*, *z'*) ∈ *set ?p* (*z*, *z'*) ∈ ℰ *S* | (*x*, *y*) ∈ *S* **by**(*auto*

```
simp add: Γ-def)
    thus ?thesis
    proof cases
      case (z z)
      hence z ∈ set p z ∈ fst ‘ ℰ S
        using y′ by(auto dest!: set-zip-leftD intro: rev-image-eqI)
      hence False by(auto dest: bypass)
      thus ?thesis ..
    qed
  next
    case False
    have ∃ e. edge Γ e (x, z) ∧ f (e, (x, z)) > 0
    proof(rule ccontr)
      assume ¬ ?thesis
     then have d-IN f (x, z) = 0 unfolding d-IN-def using currentD-outside[OF
f, of - (x, z)]
        by(force simp add: nn-integral-0-iff-AE AE-count-space not-less)
      moreover
      from xz have (x, z) ∈ S by auto
      hence (x, z) ∈ SAT Γ f by(rule orthogonal-currentD-SAT[OF ortho])
        with False have d-IN f (x, z) ≥ capacity Δ (x, z) by(auto simp add:
SAT.simps Γ-def)
      ultimately have ¬ capacity Δ (x, z) > 0 by auto
      hence ¬ edge Δ x z using capacity-pos[of (x, z)] by auto
      moreover with q have b = (x, z) by cases(auto simp add: Γ-def)
      with b have edge Δ x z by(simp add: Γ-def)
      ultimately show False by contradiction
    qed
    then obtain u where ux: edge Δ u x and xz′: edge Δ x z and uxz: edge Γ
(u, x) (x, z)
      and gt0: f ((u, x), (x, z)) > 0 by(auto simp add: Γ-def)
    have (u, x) ∈ RF° S using orthogonal-currentD-in[OF ortho, of (x, z) (u, x)]
gt0 xz
      by(fastforce intro: roofed-greaterI)
   hence ux-RF: (u, x) ∈ RF (ℰ S) and ux-ℰ: (u, x) ∉ ℰ S unfolding RF-essential
by(simp-all add: roofed-circ-def)

    from ux e have edge Γ (u, x) (x, y) by(simp add: Γ-def)
    hence path Γ (u, x) ((x, y) # ?p) ?t using p′′ ..
    from roofedD[OF ux-RF this t] ux-ℰ
    consider (x, y) ∈ S | (z) z z′ where (z, z′) ∈ set ?p (z, z′) ∈ ℰ S by auto
    thus ?thesis
    proof cases
      case (z z)
     with bypass[of z] y′ have False by(fastforce dest!: set-zip-leftD intro: rev-image-eqI)
      thus ?thesis ..
    qed
  qed
qed
```

lemma *orthogonal-flow-of-current*:
assumes *source-sink*: *source* Δ ≠ *sink* Δ
and *sink-out*: ⋀*x*. ¬ *edge* Δ (*sink* Δ) *x*
and *no-direct-edge*: ¬ *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
fix *x y*
assume *e*: *edge* Δ *x y* **and** *x* ∈ *?S* **and** *y* ∉ *?S*
then have *S*: (*x*, *y*) ∈ *S***by**(*rule orthogonal-leave-RF*)

show *?f* (*x*, *y*) = *capacity* Δ (*x*, *y*)
proof(*cases x* = *source* Δ)
case *False*
with *orthogonal-currentD-SAT*[*OF ortho S*]
have *weight* Γ (*x*, *y*) ≤ *d-IN f* (*x*, *y*) **by** *cases*(*simp-all add*: Γ*-def*)
with *False currentD-OUT-IN*[*OF f*, *of* (*x*, *y*)] *currentD-weight-IN*[*OF f*, *of* (*x*, *y*)]
show *?thesis* **by**(*simp add*: *flow-of-current-def* Γ*-def max-def*)
next
case *True*
with *orthogonal-currentD-A*[*OF ortho S*] *e currentD-weight-IN*[*OF f*, *of* (*x*, *y*)] *no-direct-edge*
show *?thesis* **by**(*auto simp add*: *flow-of-current-def* Γ*-def max-def*)
qed
next
from *sep source-sink sink-out* **have** *cut*: *cut* Δ *?S* **by**(*rule separating-cut*)

fix *x y*
assume *xy*: *edge* Δ *x y*
and *x*: *x* ∉ *?S*
and *y*: *y* ∈ *?S*
from *x* **obtain** *p* **where** *p*: *path* Δ *x p* (*sink* Δ) **and** *x'*: *x* ∉ *fst* ‘ ℰ *S*
and *bypass*: ⋀*z*. *z* ∈ *set p* ⟹ *z* ∉ *fst* ‘ ℰ *S* **by**(*auto simp add*: *roofed-def cut-of-sep-def*)
have *source*: *x* ≠ *source* Δ
proof
assume *x* = *source* Δ
have *separating-network* Δ (*fst* ‘ ℰ *S*) **using** *sep source-sink* **by**(*rule separating-network-cut-of-sep*)
from *separatingD*[*OF this p*] ‹*x* = *source* Δ› *x* **show** *False*
by(*auto dest*: *bypass intro*: *roofed-greaterI simp add*: *cut-of-sep-def*)
qed
hence *A*: (*x*, *y*) ∉ *A* Γ **by**(*simp add*: Γ*-def*)

have *f* ((*u*, *v*), *x*, *y*) = *0* **for** *u v*
proof(*cases edge* Γ (*u*, *v*) (*x*, *y*))
case *False* **with** *f* **show** *?thesis* **by**(*rule currentD-outside*)

next
case *True*
hence [*simp*]: $v = x$ **and** *ux*: *edge* Δ *u x* **by**(*auto simp add*: Γ-*def*)
have $(x, y) \in RF\ S$
proof
fix *q b*
assume *q*: *path* Γ (x, y) *q b* **and** *b*: $b \in B\ \Gamma$
define *xy* **where** $xy = (x, y)$
from *q* **have** *path* Δ (*snd xy*) (*map snd q*) (*snd b*) **unfolding** *xy-def*[*symmetric*]
by(*induction*)(*auto intro*: *rtrancl-path.intros simp add*: Γ-*def*)
with *b* **have** *path* Δ *y* (*map snd q*) (*sink* Δ) **by**(*auto simp add*: *xy-def* Γ-*def*)
from *roofedD*[*OF y*[*unfolded cut-of-sep-def*] *this*] **have** $\exists z \in set\ (y\ \#\ map\ snd\ q).\ z \in ?S$
by(*auto intro*: *roofed-greaterI simp add*: *cut-of-sep-def*)
from *split-list-last-prop*[*OF this*] **obtain** *xs z ys* **where** *decomp*: $y\ \#\ map\ snd\ q = xs\ @\ z\ \#\ ys$
and *z*: $z \in ?S$ **and** *last*: $\bigwedge z.\ z \in set\ ys \Longrightarrow z \notin ?S$ **by** *auto*
from *decomp* **obtain** x' xs' z'' ys' **where** *decomp'*: $(x', y)\ \#\ q = xs'\ @\ (z'', z)\ \#\ ys'$
and $xs = map\ snd\ xs'$ **and** *ys*: $ys = map\ snd\ ys'$ **and** x': $xs' = [] \Longrightarrow x' = x$
by(*fastforce simp add*: *Cons-eq-append-conv map-eq-append-conv*)
from *cut z* **have** *z-sink*: $z \neq sink\ \Delta$ **by** *cases*(*auto*)
then have $ys' \neq []$ **using** *rtrancl-path-last*[*OF q*] *decomp'* *b* x' *q*
by(*auto simp add*: *Cons-eq-append-conv* Γ-*def elim*: *rtrancl-path.cases*)
then obtain *w* z''' ys'' **where** ys': $ys' = (w, z''')\ \#\ ys''$ **by**(*auto simp add*: *neq-Nil-conv*)
from *q*[*THEN rtrancl-path-nth*, *of length xs'*] *decomp'* ys' x' **have** *edge* Γ (z'', z) (w, z''')
by(*auto simp add*: *Cons-eq-append-conv nth-append*)
hence *w*: $w = z$ **and** zz''': *edge* Δ *z* z''' **by**(*auto simp add*: Γ-*def*)
from ys' *ys last*[*of* z'''] **have** $z''' \notin ?S$ **by** *simp*
with zz''' *z* **have** $(z, z''') \in S$ **by**(*rule orthogonal-leave-RF*)
moreover have $(z, z''') \in set\ q$ **using** *decomp'* ys' *w* **by**(*auto simp add*: *Cons-eq-append-conv*)
ultimately show $(\exists z \in set\ q.\ z \in S) \vee (x, y) \in S$ **by** *blast*
qed
moreover
have $(u, x) \notin RF^{\circ}\ S$
proof
assume $(u, x) \in RF^{\circ}\ S$
hence *ux-RF*: $(u, x) \in RF\ (\mathcal{E}\ S)$ **and** *ux-$\mathcal{E}$*: $(u, x) \notin \mathcal{E}\ S$
unfolding *RF-essential* **by**(*simp-all add*: *roofed-circ-def*)

have $x \neq sink\ \Delta$ **using** *p xy* **by** *cases*(*auto simp add*: *sink-out*)
with *p* **have** *nNil*: $p \neq []$ **by**(*auto elim*: *rtrancl-path.cases*)
with *p* **have** *edge* Δ *x* (*hd p*) **by** *cases auto*
with *ux* **have** *edge* Γ (u, x) $(x, hd\ p)$ **by**(*simp add*: Γ-*def*)
moreover
from *p nNil* **have** *path* Γ $(x, hd\ p)$ (*zip p* (*tl p*)) (*last* $(x\ \#\ butlast\ p)$, *sink*

Δ) (**is** *path - ?x ?p ?t*)
proof(*induction*)
case (*step x y p z*)
then show *?case*
by(*cases p*)(*auto elim*: *rtrancl-path.cases intro*: *rtrancl-path.intros simp add*: Γ-*def*)
qed *simp*
ultimately have p': *path* Γ (*u*, *x*) (*?x # ?p*) *?t* **..**

have (*?x # ?p*) ! (*length p* $-$ *1*) = *?t* **using** *rtrancl-path-last*[*OF p*] *p nNil*
apply(*auto split*: *if-split-asm simp add*: *nth-Cons butlast-conv-take take-Suc-conv-app-nth not-le split*: *nat.split elim*: *rtrancl-path.cases*)
apply(*simp add*: *last-conv-nth nth-tl*)
done
moreover have *length p* $-$ *1* < *length* (*?x # ?p*) **by** *simp*
ultimately have *t*: *?t* $\in$ *B* Γ **using** *rtrancl-path-nth*[*OF p'*, *of length p* $-$ *1*]
by(*cases p*)(*simp-all add*: Γ-*def split*: *if-split-asm*)
from *roofedD*[*OF ux-RF p' t*] *ux*-$\mathcal{E}$ **consider** (*X*) (*x*, *hd p*) $\in$ $\mathcal{E}$ *S*
| (*z*) *z z'* **where** (*z*, *z'*) $\in$ *set* (*zip p* (*tl p*)) (*z*, *z'*) $\in$ $\mathcal{E}$ *S* **by** *auto*
thus *False*
proof *cases*
case *X* **with** *x'* **show** *False* **by**(*auto simp add*: *cut-of-sep-def intro*: *rev-image-eqI*)
next
case (*z z*)
with *bypass*[*of z*] **show** *False* **by**(*auto 4 3 simp add*: *cut-of-sep-def intro*: *rev-image-eqI dest*!: *set-zip-leftD*)
qed
qed
ultimately show *?thesis* **unfolding** ‹*v* = *x*› **by**(*rule orthogonal-currentD-in*[*OF ortho*])
qed
then have *d-IN f* (*x*, *y*) = *0* **unfolding** *d-IN-def*
by(*simp add*: *nn-integral-0-iff emeasure-count-space-eq-0*)
with *currentD-OUT-IN*[*OF f A*] **show** *flow-of-current* Δ *f* (*x*, *y*) = *0*
by(*simp add*: *flow-of-current-def max-def*)
qed

end

end

7.4 Avoiding antiparallel edges and self-loops

context *antiparallel-edges* **begin**

abbreviation *cut'* :: *'a vertex set* $\Rightarrow$ *'a set* **where** *cut' S* $\equiv$ *Vertex* $-$‘ *S*

lemma *cut-cut'*: *cut* Δ'' *S* $\Longrightarrow$ *cut* Δ (*cut' S*)

by(*auto simp add*: *cut.simps*)

lemma *IN-Edge*: $\mathbf{IN}_{\Delta''}$ (*Edge x y*) = (*if edge* Δ *x y then* {*Vertex x*} *else* {})
by(*auto simp add*: *incoming-def*)

lemma *OUT-Edge*: $\mathbf{OUT}_{\Delta''}$ (*Edge x y*) = (*if edge* Δ *x y then* {*Vertex y*} *else* {})
by(*auto simp add*: *outgoing-def*)

interpretation Δ'': *countable-network* Δ'' **by**(*rule* Δ''*-countable-network*)

lemma *d-IN-Edge*:
  assumes *f*: *flow* Δ'' *f*
  shows *d-IN f* (*Edge x y*) = *f* (*Vertex x*, *Edge x y*)
by(*subst d-IN-alt-def*[*OF* Δ''*.flowD-outside*[*OF f*], *of* - Δ'])(*simp-all add*: *IN-Edge nn-integral-count-space-indicator max-def* Δ''*.flowD-outside*[*OF f*])

lemma *d-OUT-Edge*:
  assumes *f*: *flow* Δ'' *f*
  shows *d-OUT f* (*Edge x y*) = *f* (*Edge x y*, *Vertex y*)
by(*subst d-OUT-alt-def*[*OF* Δ''*.flowD-outside*[*OF f*], *of* - Δ'])(*simp-all add*: *OUT-Edge nn-integral-count-space-indicator max-def* Δ''*.flowD-outside*[*OF f*])

lemma *orthogonal-cut'*:
  assumes *ortho*: *orthogonal* Δ'' *f S*
  and *f*: *flow* Δ'' *f*
  shows *orthogonal* Δ (*collect f*) (*cut' S*)
proof
  show *collect f* (*x*, *y*) = *capacity* Δ (*x*, *y*) **if** *edge*: *edge* Δ *x y* **and** *x*: $x \in$ *cut' S* **and** *y*: $y \notin$ *cut' S* **for** *x y*
  proof(*cases Edge x y* $\in$ *S*)
    case *True*
    from *y* **have** *Vertex y* $\notin$ *S* **by** *auto*
    from *orthogonalD-out*[*OF ortho* - *True this*] *edge* **show** *?thesis* **by** *simp*
  next
    case *False*
    from *x* **have** *Vertex x* $\in$ *S* **by** *auto*
    from *orthogonalD-out*[*OF ortho* - *this False*] *edge*
    have *capacity* Δ (*x*, *y*) = *d-IN f* (*Edge x y*) **by**(*simp add*: *d-IN-Edge*[*OF f*])
    also have ... = *d-OUT f* (*Edge x y*) **by**(*simp add*: *flowD-KIR*[*OF f*])
    also have ... = *f* (*Edge x y*, *Vertex y*) **using** *edge* **by**(*simp add*: *d-OUT-Edge*[*OF f*])
    finally show *?thesis* **by** *simp*
  qed

  show *collect f* (*x*, *y*) = *0* **if** *edge*: *edge* Δ *x y* **and** *x*: $x \notin$ *cut' S* **and** *y*: $y \in$ *cut' S* **for** *x y*
  proof(*cases Edge x y* $\in$ *S*)
    case *True*
    from *x* **have** *Vertex x* $\notin$ *S* **by** *auto*

from *orthogonalD-in*[*OF ortho - this True*] *edge* **have** *0* = *d-IN f* (*Edge x y*) **by**(*simp add*: *d-IN-Edge*[*OF f*])
also have ... = *d-OUT f* (*Edge x y*) **by**(*simp add*: *flowD-KIR*[*OF f*])
also have ... = *f* (*Edge x y*, *Vertex y*) **using** *edge* **by**(*simp add*: *d-OUT-Edge*[*OF f*])
finally show *?thesis* **by** *simp*
next
case *False*
from *y* **have** *Vertex y* ∈ *S* **by** *auto*
from *orthogonalD-in*[*OF ortho - False this*] *edge* **show** *?thesis* **by** *simp*
qed
qed

end

context *countable-network* **begin**

lemma *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 *countable-web* (*web-of-network* Δ) (**is** *countable-web ?Γ*)
proof
show ¬ *edge ?Γ y x* **if** *x* ∈ *A ?Γ* **for** *x y* **using** *that* **by**(*clarsimp simp add*: *source-in*)
show ¬ *edge ?Γ x y* **if** *x* ∈ *B ?Γ* **for** *x y* **using** *that* **by**(*clarsimp simp add*: *sink-out*)
show $A\ ?\Gamma \subseteq \mathbf{V}_{?\Gamma}$ **by**(*auto 4 3 dest*: *undead*)
show $A\ ?\Gamma \cap B\ ?\Gamma = \{\}$ **using** *source-sink* **by** *auto*
show ¬ *edge ?Γ x x* **for** *x* **by**(*auto simp add*: *no-loop*)
show *weight ?Γ x* = *0* **if** $x \notin \mathbf{V}_{?\Gamma}$ **for** *x* **using** *that undead*
by(*cases x*)(*auto intro*!: *capacity-outside*)
show *weight ?Γ x* ≠ ⊤ **for** *x* **using** *capacity-finite*[*of x*] **by**(*cases x*) *simp*
have $\mathbf{E}_{?\Gamma} \subseteq \mathbf{E} \times \mathbf{E}$ **by** *auto*
thus *countable* $\mathbf{E}_{?\Gamma}$ **by**(*rule countable-subset*)(*simp*)
qed

lemma *max-flow-min-cut'*:
assumes *ex-orthogonal-current*: ∃*f S*. *web-flow* (*web-of-network* Δ) *f* ∧ *separating* (*web-of-network* Δ) *S* ∧ *orthogonal-current* (*web-of-network* Δ) *f S*
and *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* Δ *f* $\wedge$ *cut* Δ *S* $\wedge$ *orthogonal* Δ *f S*
proof –
let *?Γ* = *web-of-network* Δ
from *ex-orthogonal-current* **obtain** *f S*
where *f*: *web-flow* (*web-of-network* Δ) *f*
and *S*: *separating* (*web-of-network* Δ) *S*
and *ortho*: *orthogonal-current* (*web-of-network* Δ) *f S* **by** *blast+*
let *?f* = *flow-of-current* Δ *f* **and** *?S* = *cut-of-sep* Δ *S*
from *f* **have** *flow* Δ *?f* **by**(*rule flow-flow-of-current*)
moreover have *cut* Δ *?S* **using** *S source-neq-sink sink-out* **by**(*rule separating-cut*)
moreover have *orthogonal* Δ *?f ?S* **using** *f ortho S capacity-pos source-neq-sink sink-out source-sink*
by(*rule orthogonal-flow-of-current*)
ultimately show *?thesis* **by** *blast*
qed

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* Δ $\vee$ $y =$ *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* Δ $\vee$ $y =$ *source* Δ *then 0 else capacity* Δ (x, y))
source Δ''' = *source* Δ
sink Δ''' = *sink* Δ
for *x y* **by**(*simp-all add*: Δ'''*-def*)

lemma Δ'''*-countable-network*: *countable-network* Δ'''
proof(*unfold-locales*)
have $\mathbf{E}_{\Delta'''} \subseteq \mathbf{E}$ **by** *auto*
then show *countable* $\mathbf{E}_{\Delta'''}$ **by**(*rule countable-subset*) *simp*
show *capacity* Δ''' $e = 0$ **if** $e \notin \mathbf{E}_{\Delta'''}$ **for** *e*
using *capacity-outside*[*of e*] *that* **by**(*auto split*: *if-split-asm intro*: *ccontr*)
qed(*auto simp add*: *split*: *if-split-asm*)

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*
proof –

interpret Δ''': *countable-network* Δ''' **by**(*rule* Δ'''*-countable-network*)
show *flow* Δ *f*
proof
show $f\ e \leq$ *capacity* $\Delta\ e$ **for** *e* **using** *flowD-capacity*[*OF f*, *of e*]
by(*cases e*)(*simp split*: *if-split-asm*)
show *KIR f x* **if** $x \neq$ *source* Δ $x \neq$ *sink* Δ **for** *x* **using** *flowD-KIR*[*OF f*, *of x*] *that* **by** *simp*
qed
show *cut* Δ *S* **using** *cut* **by**(*simp add*: *cut.simps*)
show *orthogonal* Δ *f S*
proof
show $f\ (x, y) =$ *capacity* $\Delta\ (x, y)$ **if** *edge*: *edge* $\Delta\ x\ y$ **and** *x*: $x \in S$ **and** *y*: $y \notin S$ **for** *x y*
proof(*cases edge* Δ''' *x y*)
case *True*
with *orthogonalD-out*[*OF ortho this x y*] **show** *?thesis* **by** *simp*
next
case *False*
from *cut y x* **have** *xy*: $y \neq$ *source* $\Delta \wedge x \neq$ *sink* Δ **by**(*cases*) *auto*
with *xy edge False* **have** *capacity* $\Delta\ (x, y) = 0$ **by** *simp*
with Δ'''*.flowD-outside*[*OF f*, *of* (x, y)] *False* **show** *?thesis* **by** *simp*
qed

show $f\ (x, y) = 0$ **if** *edge*: *edge* $\Delta\ x\ y$ **and** *x*: $x \notin S$ **and** *y*: $y \in S$ **for** *x y*
using *orthogonalD-in*[*OF ortho - x y*] Δ'''*.flowD-outside*[*OF f*, *of* (x, y)]
by(*cases edge* Δ''' *x y*)*simp-all*
qed
qed

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* $(A\ \Gamma)$
using *A-vertex countable-V* **by**(*blast intro*: *countable-subset*)

lemma *unhindered-criterion* [*rule-format*]:

assumes $\neg$ *hindered* Γ
shows $\forall X \subseteq A\ \Gamma.\ \mathit{finite}\ X \longrightarrow (\sum^{+}\ x{\in}X.\ \mathit{weight}\ \Gamma\ x) \leq (\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ \mathit{weight}\ \Gamma\ y)$
using *assms*
proof(*rule contrapos-np*)
assume $\neg$ *?thesis*
then obtain X **where** $X \in \{X.\ X \subseteq A\ \Gamma \wedge \mathit{finite}\ X \wedge (\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ \mathit{weight}\ \Gamma\ y) < (\sum^{+}\ x{\in}X.\ \mathit{weight}\ \Gamma\ x)\}$ (**is** - $\in$ *Collect ?P*)
by(*auto simp add*: *not-le*)
from *wf-eq-minimal*[*THEN iffD1*, *OF wf-finite-psubset*, *rule-format*, *OF this*, *simplified*]
obtain X **where** *X-A*: $X \subseteq A\ \Gamma$ **and** *fin-X* [*simp*]: *finite* X
and *less*: $(\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ \mathit{weight}\ \Gamma\ y) < (\sum^{+}\ x{\in}X.\ \mathit{weight}\ \Gamma\ x)$
and *minimal*: $\bigwedge X'.\ X' \subset X \Longrightarrow (\sum^{+}\ x{\in}X'.\ \mathit{weight}\ \Gamma\ x) \leq (\sum^{+}\ y{\in}\mathbf{E}\ ``\ X'.\ \mathit{weight}\ \Gamma\ y)$
by(*clarsimp simp add*: *not-less*)(*meson finite-subset order-trans psubset-imp-subset*)
have *nonempty*: $X \neq \{\}$ **using** *less* **by** *auto*
then obtain *xx* **where** *xx*: $xx \in X$ **by** *auto*
define f **where**
$f\ x = (\mathit{if}\ x = xx\ \mathit{then}\ (\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ \mathit{weight}\ \Gamma\ y) - (\sum^{+}\ x{\in}X - \{xx\}.\ \mathit{weight}\ \Gamma\ x)\ \mathit{else}\ \mathit{if}\ x \in X\ \mathit{then}\ \mathit{weight}\ \Gamma\ x\ \mathit{else}\ 0)$ **for** x
define g **where**
$g\ y = (\mathit{if}\ y \in \mathbf{E}\ ``\ X\ \mathit{then}\ \mathit{weight}\ \Gamma\ y\ \mathit{else}\ 0)$ **for** y
define E' **where** $E' \equiv \mathbf{E} \cap X \times \mathit{UNIV}$
have *Xxx*: $X - \{xx\} \subset X$ **using** *xx* **by** *blast*
have E [*simp*]: $E'\ ``\ X' = \mathbf{E}\ ``\ X'$ **if** $X' \subseteq X$ **for** X' **using** *that* **by**(*auto simp add*: *E′-def*)
have *in-E′*: $(x,\ y) \in E' \longleftrightarrow x \in X \wedge (x,\ y) \in \mathbf{E}$ **for** $x\ y$ **by**(*auto simp add*: *E′-def*)

have $(\sum^{+}\ x{\in}X.\ f\ x) = (\sum^{+}\ x{\in}X - \{xx\}.\ f\ x) + (\sum^{+}\ x{\in}\{xx\}.\ f\ x)$ **using** *xx*
by(*auto simp add*: *nn-integral-count-space-indicator nn-integral-add*[*symmetric*] *simp del*: *nn-integral-indicator-singleton intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots = (\sum^{+}\ x{\in}X - \{xx\}.\ \mathit{weight}\ \Gamma\ x) + ((\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ \mathit{weight}\ \Gamma\ y) - (\sum^{+}\ x{\in}X - \{xx\}.\ \mathit{weight}\ \Gamma\ x))$
by(*rule arg-cong2*[**where** f=(+)])(*auto simp add*: *f-def xx nn-integral-count-space-indicator intro*!: *nn-integral-cong*)
also have $\ldots = (\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ g\ y)$ **using** *minimal*[*OF Xxx*] *xx*
by(*subst add-diff-eq-iff-ennreal*[*THEN iffD2*])(*fastforce simp add*: *g-def*[*abs-def*] *nn-integral-count-space-indicator intro*!: *nn-integral-cong intro*: *nn-integral-mono elim*: *order-trans split*: *split-indicator*)+
finally have *sum-eq*: $(\sum^{+}\ x{\in}X.\ f\ x) = (\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ g\ y)$ **.**

have $(\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ \mathit{weight}\ \Gamma\ y) = (\sum^{+}\ y{\in}\mathbf{E}\ ``\ X.\ g\ y)$
by(*auto simp add*: *nn-integral-count-space-indicator g-def intro*!: *nn-integral-cong*)
then have *fin*: $\ldots \neq \top$ **using** *less* **by** *auto*

have *fin2*: $(\sum^{+}\ x{\in}X'.\ \mathit{weight}\ \Gamma\ x) \neq \top$ **if** $X' \subset X$ **for** X'
proof −

have $(\sum^+ x{\in}\mathbf{E}$ *‘‘* $X'.$ *weight* Γ $x) \leq (\sum^+ x{\in}\mathbf{E}$ *‘‘* $X.$ *weight* Γ $x)$ **using** *that*
by(*auto 4 3 simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator split-indicator-asm*)
then show *?thesis* **using** *minimal*[*OF that*] *less* **by**(*auto simp add*: *top-unique*)
qed

have $f\ xx = (\sum^+ y{\in}\mathbf{E}$ *‘‘* $X.$ *weight* Γ $y) - (\sum^+ x{\in}X - \{xx\}.$ *weight* Γ $x)$ **by** (*simp add*: *f-def*)
also have $\ldots < (\sum^+ x{\in}X.$ *weight* Γ $x) - (\sum^+ x{\in}X - \{xx\}.$ *weight* Γ $x)$ **using** *less fin2*[*OF Xxx*] *minimal*[*OF Xxx*]
by(*subst minus-less-iff-ennreal*)(*fastforce simp add*: *less-top*[*symmetric*] *nn-integral-count-space-indicator diff-add-self-ennreal intro*: *nn-integral-mono elim*: *order-trans split*: *split-indicator*)+
also have $\ldots = (\sum^+ x{\in}\{xx\}.$ *weight* Γ $x)$ **using** *fin2*[*OF Xxx*] *xx*
apply(*simp add*: *nn-integral-count-space-indicator del*: *nn-integral-indicator-singleton*)
apply(*subst nn-integral-diff* [*symmetric*])
apply(*auto simp add*: *AE-count-space split*: *split-indicator simp del*: *nn-integral-indicator-singleton intro*!: *nn-integral-cong*)
done
also have $\ldots =$ *weight* Γ *xx* **by**(*simp add*: *nn-integral-count-space-indicator*)
finally have *fxx*: $f\ xx <$ *weight* Γ *xx* **.**

have *le*: $(\sum^+ x{\in}X'.\ f\ x) \leq (\sum^+ y{\in}\mathbf{E}$ *‘‘* $X'.\ g\ y)$ **if** $X' \subseteq X$ **for** X'
proof(*cases* $X' = X$)
case *True*
then show *?thesis* **using** *sum-eq* **by** *simp*
next
case *False*
hence X': $X' \subset X$ **using** *that* **by** *blast*
have $(\sum^+ x{\in}X'.\ f\ x) = (\sum^+ x{\in}X' - \{xx\}.\ f\ x) + (\sum^+ x{\in}\{xx\}.\ f\ x\ *$ *indicator* $X'\ xx)$
by(*auto simp add*: *nn-integral-count-space-indicator nn-integral-add*[*symmetric*] *simp del*: *nn-integral-indicator-singleton intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots \leq (\sum^+ x{\in}X' - \{xx\}.\ f\ x) + (\sum^+ x{\in}\{xx\}.$ *weight* $\Gamma\ x\ *$ *indicator* $X'\ xx)$ **using** *fxx*
by(*intro add-mono*)(*auto split*: *split-indicator simp add*: *nn-integral-count-space-indicator*)
also have $\ldots = (\sum^+ x{\in}X'.$ *weight* Γ $x)$ **using** *xx that*
by(*auto simp add*: *nn-integral-count-space-indicator nn-integral-add*[*symmetric*] *f-def simp del*: *nn-integral-indicator-singleton intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots \leq (\sum^+ y{\in}\mathbf{E}$ *‘‘* $X'.$ *weight* Γ $y)$ **by**(*rule minimal*[*OF X'*])
also have $\ldots = (\sum^+ y{\in}\mathbf{E}$ *‘‘* $X'.\ g\ y)$ **using** *that*
by(*auto 4 3 intro*!: *nn-integral-cong simp add*: *g-def Image-iff*)
finally show *?thesis* **.**
qed

have *countable X* **using** *X-A A-vertex countable-V* **by**(*blast intro*: *countable-subset*)
moreover have $\mathbf{E}$ *‘‘* $X \subseteq \mathbf{V}$ **by**(*auto simp add*: *vertex-def*)
with *countable-V* **have** *countable* ($\mathbf{E}$ *‘‘* X) **by**(*blast intro*: *countable-subset*)
moreover have $E' \subseteq X \times \mathbf{E}$ *‘‘* X **by**(*auto simp add*: *E'-def*)
ultimately obtain h' **where** *h'-dom*: $\bigwedge x\ y.\ 0 < h'\ x\ y \Longrightarrow (x, y) \in E'$

and h'-*fin*: $\bigwedge x\ y.\ h'\ x\ y \neq \top$
and h'-*f*: $\bigwedge x.\ x \in X \Longrightarrow (\sum^{+}\ y{\in}E'\ ``\ X.\ h'\ x\ y) = f\ x$
and h'-*g*: $\bigwedge y.\ y \in E'\ ``\ X \Longrightarrow (\sum^{+}\ x{\in}X.\ h'\ x\ y) = g\ y$
using *bounded-matrix-for-marginals-ennreal*[**where** $f{=}f$ **and** $g{=}g$ **and** $A{=}X$ **and** $B{=}E'\ ``\ X$ **and** $R{=}E'$ **and** *thesis*=*thesis*] *sum-eq fin le*
by(*auto*)

have h'-*outside*: $(x,\ y) \notin E' \Longrightarrow h'\ x\ y = 0$ **for** $x\ y$ **using** h'-*dom*[*of* $x\ y$] *not-gr-zero* **by**(*fastforce*)

define h **where** $h = (\lambda(x,\ y).\ \mathit{if}\ x \in X \wedge \mathit{edge}\ \Gamma\ x\ y\ \mathit{then}\ h'\ x\ y\ \mathit{else}\ 0)$
have h-*OUT*: *d-OUT* $h\ x = (\mathit{if}\ x \in X\ \mathit{then}\ f\ x\ \mathit{else}\ 0)$ **for** x
by(*auto 4 3 simp add*: *h-def d-OUT-def* h'-*f*[*symmetric*] E'-*def nn-integral-count-space-indicator intro*!: *nn-integral-cong intro*: h'-*outside split*: *split-indicator*)
have h-*IN*: *d-IN* $h\ y = (\mathit{if}\ y \in \mathbf{E}\ ``\ X\ \mathit{then}\ \mathit{weight}\ \Gamma\ y\ \mathit{else}\ 0)$ **for** y **using** h'-*g*[*of* y, *symmetric*]
by(*auto 4 3 simp add*: *h-def d-IN-def g-def nn-integral-count-space-indicator nn-integral-0-iff-AE in-*E' *intro*!: *nn-integral-cong intro*: h'-*outside split*: *split-indicator split-indicator-asm*)

have h: *current* Γ h
proof
show *d-OUT* $h\ x \leq$ *weight* $\Gamma\ x$ **for** x **using** *fxx* **by**(*auto simp add*: *h-OUT f-def*)
show *d-IN* $h\ y \leq$ *weight* $\Gamma\ y$ **for** y **by**(*simp add*: *h-IN*)
show $h\ e = 0$ **if** $e \notin \mathbf{E}$ **for** e **using** *that* **by**(*cases* e)(*auto simp add*: *h-def*)
qed

have *separating* Γ (*TER* h)
proof
fix $x\ y\ p$
assume x: $x \in A\ \Gamma$ **and** y: $y \in B\ \Gamma$ **and** p: *path* $\Gamma\ x\ p\ y$
then obtain [*simp*]: $p = [y]$ **and** *xy*: $(x,\ y) \in \mathbf{E}$ **using** *disjoint*
by $-$(*erule rtrancl-path.cases*; *auto dest*: *bipartite-E*)+
show $(\exists z{\in}\mathit{set}\ p.\ z \in \mathit{TER}\ h) \vee x \in \mathit{TER}\ h$
proof(*rule disjCI*)
assume $x \notin$ *TER* h
hence $x \in X$ **using** x **by**(*auto simp add*: *SAT.simps SINK.simps h-OUT split*: *if-split-asm*)
hence $y \in$ *TER* h **using** *xy currentD-OUT*[*OF* $h\ y$] **by**(*auto simp add*: *SAT.simps h-IN SINK.simps*)
thus $\exists z{\in}\mathit{set}\ p.\ z \in \mathit{TER}\ h$ **by** *simp*
qed
qed
then have w: *wave* Γ h **using** h **..**

have $xx \in A\ \Gamma$ **using** *xx X-A* **by** *blast*
moreover have $xx \notin \mathcal{E}$ (*TER* h)
proof

assume $xx \in \mathcal{E}$ (*TER h*)
then obtain *p y* **where** *y*: $y \in B\ \Gamma$ **and** *p*: *path* Γ *xx p y*
and *bypass*: $\bigwedge z.\ [\![\ xx \neq y;\ z \in set\ p\ ]\!] \Longrightarrow z = xx \vee z \notin TER\ h$
by(*rule* $\mathcal{E}$-*E*) *auto*
from *p* **obtain** [*simp*]: $p = [y]$ **and** *xy*: *edge* Γ *xx y* **and** *neq*: $xx \neq y$ **using** *disjoint X-A xx y*
by −(*erule rtrancl-path.cases*; *auto dest*: *bipartite-E*)+
from *neq bypass*[*of y*] **have** $y \notin TER\ h$ **by** *simp*
moreover from *xy xx currentD-OUT*[*OF h y*] **have** $y \in TER\ h$
by(*auto simp add*: *SAT.simps h-IN SINK.simps*)
ultimately show *False* **by** *contradiction*
qed
moreover have *d-OUT h xx* < *weight* Γ *xx* **using** *fxx xx* **by**(*simp add*: *h-OUT*)
ultimately have *hindrance* Γ *h* **..**
then show *hindered* Γ **using** *h w* **..**
qed

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*}
using *assms* **unfolding** *nn-integral-count-space-nat suminf-eq-SUP SUP-eq-top-iff*
by(*auto*)

lemma *One-le-of-nat-ennreal*: (*1* :: *ennreal*) ≤ *of-nat x* ⟷ *1* ≤ *x*
by (*metis of-nat-le-iff of-nat-1*)

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 ¬ *hindered* Γ
shows *linkable* Γ
proof(*cases* $A\ \Gamma = \{\}$)
case *True*
hence *linkage* Γ (λ-. *0*) **by**(*auto simp add*: *linkage.simps*)
moreover have *web-flow* Γ (λ-. *0*) **by**(*auto simp add*: *web-flow.simps*)
ultimately show *?thesis* **by** *blast*
next
case *nonempty*: *False*
define *A-n* :: *nat* ⇒ *'v set* **where** *A-n n* = *from-nat-into* ($A\ \Gamma$) ‘ {..*n*} **for** *n*
have *fin-A-n* [*simp*]: *finite* (*A-n n*) **for** *n* **by**(*simp add*: *A-n-def*)
have *A-n-A*: *A-n n* ⊆ $A\ \Gamma$ **for** *n* **by**(*auto simp add*: *A-n-def from-nat-into*[*OF*

nonempty])

have *countable2*: *countable* (**E** `` *A-n n*) **for** *n* **using** *countable-V*
by(*rule countable-subset*[*rotated*])(*auto simp add*: *vertex-def*)

have $\exists$ *Y2*. $\forall$ *n*. $\forall$ *X* $\subseteq$ *A-n n*. *Y2 n X* $\subseteq$ **E** `` *X* $\wedge$ ($\sum^+$ *x*$\in$*X*. *weight* Γ *x*) $\leq$ ($\sum^+$ *y*$\in$*Y2 n X*. *weight* Γ *y*) $\wedge$ ($\sum^+$ *y*$\in$*Y2 n X*. *weight* Γ *y*) $\neq$ $\top$
proof(*rule choice strip ex-simps*(*6*)[*THEN iffD2*])+
fix *n X*
assume *X*: *X* $\subseteq$ *A-n n*
then have [*simp*]: *finite X* **by**(*rule finite-subset*) *simp*
have *X-count*: *countable* (**E** `` *X*) **using** *countable2*
by(*rule countable-subset*[*rotated*])(*rule Image-mono*[*OF order-refl X*])

show $\exists$ *Y*. *Y* $\subseteq$ **E** `` *X* $\wedge$ ($\sum^+$ *x*$\in$*X*. *weight* Γ *x*) $\leq$ ($\sum^+$ *y*$\in$*Y*. *weight* Γ *y*) $\wedge$ ($\sum^+$ *y*$\in$*Y*. *weight* Γ *y*) $\neq$ $\top$ (**is** *Ex ?P*)
proof(*cases* ($\sum^+$ *y*$\in$**E** `` *X*. *weight* Γ *y*) = $\top$)
case *True*
define *Y'* **where** *Y'* = *to-nat-on* (**E** `` *X*) ` (**E** `` *X*)
have *inf*: *infinite* (**E** `` *X*) **using** *True*
by(*intro notI*)(*auto simp add*: *nn-integral-count-space-finite*)
then have *Y'*: *Y'* = *UNIV* **using** *X-count* **by**(*auto simp add*: *Y'-def intro*!: *image-to-nat-on*)
have ($\sum^+$ *y*$\in$**E** `` *X*. *weight* $\Gamma$ *y*) = ($\sum^+$ *y*$\in$*from-nat-into* (**E** `` *X*) ` *Y'*. *weight* Γ *y* $*$ *indicator* (**E** `` *X*) *y*)
using *X-count*
by(*auto simp add*: *nn-integral-count-space-indicator Y'-def image-image intro*!: *nn-integral-cong from-nat-into-to-nat-on*[*symmetric*] *rev-image-eqI split*: *split-indicator*)
also have ... = ($\sum^+$ *y*. *weight* Γ (*from-nat-into* (**E** `` *X*) *y*) $*$ *indicator* (**E** `` *X*) (*from-nat-into* (**E** `` *X*) *y*))
using *X-count inf* **by**(*subst nn-integral-count-space-reindex*)(*auto simp add*: *inj-on-def Y'*)
finally have ... = $\top$ **using** *True* **by** *simp*
from *nn-integral-count-space-top-approx*[*OF this*, *of sum* (*weight* Γ) *X*]
obtain *yy* **where** *yy*: *sum* (*weight* Γ) *X* < ($\sum$ *y*<*yy*. *weight* Γ (*from-nat-into* (**E** `` *X*) *y*) $*$ *indicator* (**E** `` *X*) (*from-nat-into* (**E** `` *X*) *y*))
by(*auto simp add*: *less-top*[*symmetric*])
define *Y* **where** *Y* = *from-nat-into* (**E** `` *X*) ` {..<*yy*} $\cap$ **E** `` *X*
have [*simp*]: *finite Y* **by**(*simp add*: *Y-def*)
have ($\sum^+$ *x*$\in$*X*. *weight* Γ *x*) = *sum* (*weight* Γ) *X* **by**(*simp add*: *nn-integral-count-space-finite*)
also have ... $\leq$ ($\sum$ *y*<*yy*. *weight* Γ (*from-nat-into* (**E** `` *X*) *y*) $*$ *indicator* (**E** `` *X*) (*from-nat-into* (**E** `` *X*) *y*))
using *yy* **by** *simp*
also have ... = ($\sum$ *y* $\in$ *from-nat-into* (**E** `` *X*) ` {..<*yy*}. *weight* $\Gamma$ *y* $*$ *indicator* (**E** `` *X*) *y*)
using *X-count inf* **by**(*subst sum.reindex*)(*auto simp add*: *inj-on-def*)
also have ... = ($\sum$ *y* $\in$ *Y*. *weight* Γ *y*) **by**(*auto intro*!: *sum.cong simp add*: *Y-def*)
also have ... = ($\sum^+$ *y*$\in$*Y*. *weight* Γ *y*) **by**(*simp add*: *nn-integral-count-space-finite*)

also have $Y \subseteq \mathbf{E}$ `` X **by**(*simp add*: *Y-def*)
moreover have $(\sum^+ y{\in}Y.\ weight\ \Gamma\ y) \neq \top$ **by**(*simp add*: *nn-integral-count-space-finite*)
ultimately show *?thesis* **by** *blast*
next
case *False*
with *unhindered-criterion*[*OF assms*, *of X*] *X A-n-A*[*of n*] **have** *?P* ($\mathbf{E}$ `` X) **by** *auto*
then show *?thesis* **..**
qed
qed
then obtain *Y2*
where *Y2-A*: $Y2\ n\ X \subseteq \mathbf{E}$ `` X
and *le*: $(\sum^+ x{\in}X.\ weight\ \Gamma\ x) \leq (\sum^+ y{\in}Y2\ n\ X.\ weight\ \Gamma\ y)$
and *finY2*: $(\sum^+ y{\in}Y2\ n\ X.\ weight\ \Gamma\ y) \neq \top$ **if** $X \subseteq A\text{-}n\ n$ **for** *n X* **by** *iprover*
define *Y* **where** $Y\ n = (\bigcup X \in Pow\ (A\text{-}n\ n).\ Y2\ n\ X)$ **for** *n*
define *s* **where** $s\ n = (\sum^+ y{\in}Y\ n.\ weight\ \Gamma\ y)$ **for** *n*
have *Y-vertex*: $Y\ n \subseteq \mathbf{V}$ **for** *n* **by**(*auto 4 3 simp add*: *Y-def vertex-def dest*!: *Y2-A*[*of - n*])
have *Y-B*: $Y\ n \subseteq B\ \Gamma$ **for** *n* **unfolding** *Y-def* **by**(*auto dest*!: *Y2-A*[*of - n*] *dest*: *bipartite-E*)

have *s-top* [*simp*]: $s\ n \neq \top$ **for** *n*
proof −
have $[\![x \in Y2\ n\ X;\ X \subseteq A\text{-}n\ n]\!] \Longrightarrow Suc\ 0 \leq card\ \{X.\ X \subseteq A\text{-}n\ n \wedge x \in Y2\ n\ X\}$ **for** *x X*
by(*subst card-le-Suc-iff*)(*auto intro*!: *exI*[**where** $x{=}X$] *exI*[**where** $x{=}\{X.\ X \subseteq A\text{-}n\ n \wedge x \in Y2\ n\ X\} - \{X\}$])
then have $(\sum^+ y{\in}Y\ n.\ weight\ \Gamma\ y) \leq (\sum^+ y{\in}Y\ n.\ \sum X{\in}Pow\ (A\text{-}n\ n).\ weight\ \Gamma\ y * indicator\ (Y2\ n\ X)\ y)$
by(*intro nn-integral-mono*)(*auto simp add*: *Y-def One-le-of-nat-ennreal intro*!: *mult-right-mono*[*of 1* :: *ennreal*, *simplified*])
also have $\ldots = (\sum X{\in}Pow\ (A\text{-}n\ n).\ \sum^+ y{\in}Y\ n.\ weight\ \Gamma\ y * indicator\ (Y2\ n\ X)\ y)$
by(*subst nn-integral-sum*) *auto*
also have $\ldots = (\sum X{\in}Pow\ (A\text{-}n\ n).\ \sum^+ y{\in}Y2\ n\ X.\ weight\ \Gamma\ y)$
by(*auto intro*!: *sum.cong nn-integral-cong simp add*: *nn-integral-count-space-indicator Y-def split*: *split-indicator*)
also have $\ldots < \top$ **by**(*simp add*: *less-top*[*symmetric*] *finY2*)
finally show *?thesis* **by**(*simp add*: *less-top s-def*)
qed

define $f :: nat \Rightarrow {'}v\ option \Rightarrow real$
where $f\ n\ xo = (case\ xo\ of\ Some\ x \Rightarrow if\ x \in A\text{-}n\ n\ then\ enn2real\ (weight\ \Gamma\ x)\ else\ 0$
$\mid None \Rightarrow enn2real\ (s\ n - sum\ (weight\ \Gamma)\ (A\text{-}n\ n)))$ **for** *n xo*
define $g :: nat \Rightarrow {'}v \Rightarrow real$
where $g\ n\ y = enn2real\ (weight\ \Gamma\ y * indicator\ (Y\ n)\ y)$ **for** *n y*
define $R :: nat \Rightarrow ({'}v\ option \times {'}v)\ set$
where $R\ n = map\text{-}prod\ Some\ id$ ` $(\mathbf{E} \cap A\text{-}n\ n \times Y\ n) \cup \{None\} \times Y\ n$ **for** *n*

define $A\text{-}n'$ **where** $A\text{-}n'\ n = Some\ `\ A\text{-}n\ n \cup \{None\}$ **for** n

have *f-simps*:
$f\ n\ (Some\ x) = (if\ x \in A\text{-}n\ n\ then\ enn2real\ (weight\ \Gamma\ x)\ else\ 0)$
$f\ n\ None = enn2real\ (s\ n - sum\ (weight\ \Gamma)\ (A\text{-}n\ n))$
for $n\ x$ **by**(*simp-all add*: *f-def*)

have *g-s*: $(\sum^+ y{\in}Y\ n.\ g\ n\ y) = s\ n$ **for** n
by(*auto simp add*: *s-def g-def ennreal-enn2real-if intro*!: *nn-integral-cong*)

have $(\sum^+ x{\in}A\text{-}n'\ n.\ f\ n\ x) = (\sum^+ x{\in}Some`A\text{-}n\ n.\ weight\ \Gamma\ (the\ x)) + (\sum^+ x{\in}\{None\}.\ f\ n\ x)$ **for** n
by(*auto simp add*: *nn-integral-count-space-indicator nn-integral-add*[*symmetric*] *f-simps A-n′-def ennreal-enn2real-if simp del*: *nn-integral-indicator-singleton intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots\ n = sum\ (weight\ \Gamma)\ (A\text{-}n\ n) + (s\ n - sum\ (weight\ \Gamma)\ (A\text{-}n\ n))$ **for** n
by(*subst nn-integral-count-space-reindex*)(*auto simp add*: *nn-integral-count-space-finite f-simps ennreal-enn2real-if*)
also have $\ldots\ n = s\ n$ **for** n **using** *le*[*OF order-refl*, *of n*]
by(*simp add*: *s-def nn-integral-count-space-finite*)(*auto elim*!: *order-trans simp add*: *nn-integral-count-space-indicator Y-def intro*!: *nn-integral-mono split*: *split-indicator*)
finally have *sum-eq*: $(\sum^+ x{\in}A\text{-}n'\ n.\ f\ n\ x) = (\sum^+ y{\in}Y\ n.\ g\ n\ y)$ **for** n **using** *g-s* **by** *simp*

have $\exists\, h'.\ \forall\, n.\ (\forall\, x\ y.\ (x,\ y) \notin R\ n \longrightarrow h'\ n\ x\ y = 0) \wedge (\forall\, x\ y.\ h'\ n\ x\ y \neq \top) \wedge (\forall\, x{\in}A\text{-}n'\ n.\ (\sum^+ y{\in}Y\ n.\ h'\ n\ x\ y) = f\ n\ x) \wedge (\forall\, y{\in}Y\ n.\ (\sum^+ x{\in}A\text{-}n'\ n.\ h'\ n\ x\ y) = g\ n\ y)$
(**is** $Ex\ (\lambda h'.\ \forall\, n.\ ?Q\ n\ (h'\ n))$)
proof(*rule choice allI*)+
fix n
note *sum-eq*
moreover have $(\sum^+ y{\in}Y\ n.\ g\ n\ y) \neq \top$ **using** *g-s* **by** *simp*
moreover have *le-fg*: $(\sum^+ x{\in}X.\ f\ n\ x) \leq (\sum^+ y{\in}R\ n\ ``\ X.\ g\ n\ y)$ **if** $X \subseteq A\text{-}n'\ n$ **for** X
proof(*cases* $None \in X$)
case *True*
have $(\sum^+ x{\in}X.\ f\ n\ x) \leq (\sum^+ x{\in}A\text{-}n'\ n.\ f\ n\ x)$ **using** *that*
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
also have $\ldots = (\sum^+ y{\in}Y\ n.\ g\ n\ y)$ **by**(*simp add*: *sum-eq*)
also have $R\ n\ ``\ X = Y\ n$ **using** *True* **by**(*auto simp add*: *R-def*)
ultimately show *?thesis* **by** *simp*
next
case *False*
then have $*$: $Some\ `\ (the\ `\ X) = X$
by(*auto simp add*: *image-image*)(*metis* (*no-types*, *lifting*) *image-iff notin-range-Some option.sel option.collapse*)+
from *False that* **have** X: $the\ `\ X \subseteq A\text{-}n\ n$ **by**(*auto simp add*: *A-n′-def*)

from $*$ **have** $(\sum^+ x{\in}X.\ f\ n\ x) = (\sum^+ x{\in}Some\ `\ (the\ `\ X).\ f\ n\ x)$ **by** *simp*
also have $\ldots = (\sum^+ x{\in}the\ `\ X.\ f\ n\ (Some\ x))$ **by**(*rule nn-integral-count-space-reindex*) *simp*
also have $\ldots = (\sum^+ x{\in}the\ `\ X.\ weight\ \Gamma\ x)$ **using** *that False*
by(*auto 4 3intro*!: *nn-integral-cong simp add*: *f-simps A-n′-def ennreal-enn2real-if*)
also have $\ldots \leq (\sum^+ y{\in}Y2\ n\ (the\ `\ X).\ weight\ \Gamma\ y)$ **using** *False that*
by(*intro le*)(*auto simp add*: *A-n′-def*)
also have $\ldots \leq (\sum^+ y{\in}R\ n\ ``\ X.\ weight\ \Gamma\ y)$ **using** *False Y2-A*[*of the ‘ X n*] *X*
by(*auto simp add*: *A-n′-def nn-integral-count-space-indicator R-def Image-iff Y-def intro*: *rev-image-eqI intro*!: *nn-integral-mono split*: *split-indicator*)
(*drule* (*1*) *subsetD*; *clarify*; *drule* (*1*) *bspec*; *auto 4 3 intro*: *rev-image-eqI*)
also have $\ldots = (\sum^+ y{\in}R\ n\ ``\ X.\ g\ n\ y)$
by(*auto intro*!: *nn-integral-cong simp add*: *R-def g-def ennreal-enn2real-if*)
finally show *?thesis* **.**
qed
moreover have *countable* (A-n' n) **by**(*simp add*: *A-n′-def countable-finite*)
moreover have *countable* (*Y2 n X*) **if** $X \subseteq A\text{-}n\ n$ **for** X **using** *Y2-A*[*OF that*]
by(*rule countable-subset*)(*rule countable-subset*[*OF - countable-V*]; *auto simp add*: *vertex-def*)
then have *countable* (*Y n*) **unfolding** *Y-def*
by(*intro countable-UN*)(*simp-all add*: *countable-finite*)
moreover have $R\ n \subseteq A\text{-}n'\ n \times Y\ n$ **by**(*auto simp add*: *R-def A-n′-def*)
ultimately obtain h' **where** $\bigwedge x\ y.\ 0 < h'\ x\ y \Longrightarrow (x,\ y) \in R\ n$ $\bigwedge x\ y.\ h'\ x\ y \neq \top$
$\bigwedge x.\ x \in A\text{-}n'\ n \Longrightarrow (\sum^+ y{\in}Y\ n.\ h'\ x\ y) = (f\ n\ x)$ $\bigwedge y.\ y \in Y\ n \Longrightarrow (\sum^+ x{\in}A\text{-}n'\ n.\ h'\ x\ y) = g\ n\ y$
by(*rule bounded-matrix-for-marginals-ennreal*) *blast+*
hence *?Q n h′* **by**(*auto*)(*use not-gr-zero* **in** *blast*)
thus *Ex* (*?Q n*) **by** *blast*
qed
then obtain h' **where** *dom-h′*: $\bigwedge x\ y.\ (x,\ y) \notin R\ n \Longrightarrow h'\ n\ x\ y = 0$
and *fin-h′* [*simp*]: $\bigwedge x\ y.\ h'\ n\ x\ y \neq \top$
and *h′-f*: $\bigwedge x.\ x \in A\text{-}n'\ n \Longrightarrow (\sum^+ y{\in}Y\ n.\ h'\ n\ x\ y) = f\ n\ x$
and *h′-g*: $\bigwedge y.\ y \in Y\ n \Longrightarrow (\sum^+ x{\in}A\text{-}n'\ n.\ h'\ n\ x\ y) = g\ n\ y$
for *n* **by** *blast*

define $h :: nat \Rightarrow {}'v \times {}'v \Rightarrow real$
where $h\ n = (\lambda(x,\ y).\ if\ x \in A\text{-}n\ n \wedge y \in Y\ n\ then\ enn2real\ (h'\ n\ (Some\ x)\ y)\ else\ 0)$ **for** *n*
have *h-nonneg*: $0 \leq h\ n\ xy$ **for** *n xy* **by**(*simp add*: *h-def split-def*)
have *h-notB*: $h\ n\ (x,\ y) = 0$ **if** $y \notin B\ \Gamma$ **for** *n x y* **using** *Y-B*[*of n*] *that* **by**(*auto simp add*: *h-def*)
have *h-le-weight2*: $h\ n\ (x,\ y) \leq weight\ \Gamma\ y$ **for** *n x y*
proof(*cases* $x \in A\text{-}n\ n \wedge y \in Y\ n$)
case *True*
have $h'\ n\ (Some\ x)\ y \leq (\sum^+ x{\in}A\text{-}n'\ n.\ h'\ n\ x\ y)$
by(*rule nn-integral-ge-point*)(*auto simp add*: *A-n′-def True*)
also have $\ldots \leq weight\ \Gamma\ y$ **using** *h′-g*[*of y n*] *True* **by**(*simp add*: *g-def*

ennreal-enn2real-if)
finally show *?thesis* **using** *True* **by**(*simp add*: *h-def ennreal-enn2real-if*)
qed(*auto simp add*: *h-def*)
have *h-OUT*: *d-OUT* (*h n*) *x* = (*if x* ∈ *A-n n then weight* Γ *x else 0*) **for** *n x*
using *h′-f*[*of Some x n*, *symmetric*]
by(*auto simp add*: *h-def d-OUT-def A-n′-def f-simps ennreal-enn2real-if nn-integral-count-space-indicator intro*!: *nn-integral-cong*)
have *h-IN*: *d-IN* (*h n*) *y* = (*if y* ∈ *Y n then enn2real* (*weight* Γ *y* − *h′ n None y*) *else 0*) **for** *n y*
proof(*cases y* ∈ *Y n*)
case *True*
have *d-IN* (*h n*) *y* = (∑⁺ *x*∈*Some* ‘ *A-n n*. *h′ n x y*)
by(*subst nn-integral-count-space-reindex*)
(*auto simp add*: *d-IN-def h-def nn-integral-count-space-indicator ennreal-enn2real-if R-def intro*!: *nn-integral-cong dom-h′ split*: *split-indicator*)
also have ... = (∑⁺ *x*∈*A-n′ n*. *h′ n x y*) − (∑⁺ *x*∈{*None*}. *h′ n x y*)
apply(*simp add*: *nn-integral-count-space-indicator del*: *nn-integral-indicator-singleton*)
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space A-n′-def nn-integral-count-space-indicator split*: *split-indicator intro*!: *nn-integral-cong*)
done
also have ... = *g n y* − *h′ n None y* **using** *h′-g*[*OF True*] **by**(*simp add*: *nn-integral-count-space-indicator*)
finally show *?thesis* **using** *True* **by**(*simp add*: *g-def ennreal-enn2real-if*)
qed(*auto simp add*: *d-IN-def ennreal-enn2real-if nn-integral-0-iff-AE AE-count-space h-def g-def*)

let *?Q* = **V** × **V**

have *bounded* (*range* (λ*n*. *h n xy*)) **if** *xy* ∈ *?Q* **for** *xy* **unfolding** *bounded-def dist-real-def*
proof(*rule exI strip*|*erule imageE*|*hypsubst*)+
fix *n*
obtain *x y* **where** [*simp*]: *xy* = (*x*, *y*) **by**(*cases xy*)
have *h n* (*x*, *y*) ≤ *d-OUT* (*h n*) *x* **unfolding** *d-OUT-def* **by**(*rule nn-integral-ge-point*) *simp*
also have ... ≤ *weight* Γ *x* **by**(*simp add*: *h-OUT*)
finally show |*0* − *h n xy*| ≤ *enn2real* (*weight* Γ (*fst xy*))
by(*simp add*: *h-nonneg*)(*metis enn2real-ennreal ennreal-cases ennreal-le-iff weight-finite*)
qed
moreover have *countable ?Q* **using** *countable-V* **by**(*simp*)
ultimately obtain *k* **where** *k*: *strict-mono k*
and *conv*: ⋀*xy*. *xy* ∈ *?Q* ⟹ *convergent* (λ*n*. *h* (*k n*) *xy*)
by(*rule convergent-bounded-family*) *blast*+

have *h-outside*: *h n xy* = *0* **if** *xy* ∉ *?Q* **for** *xy n* **using** *that A-n-A*[*of n*] *A-vertex Y-vertex*
by(*auto simp add*: *h-def split*: *prod.split*)

have *h-outside-AB*: *h n* $(x, y) = 0$ **if** $x \notin A\ \Gamma \vee y \notin B\ \Gamma$ **for** *n x y*
using *that A-n-A*[*of n*] *Y-B*[*of n*] **by**(*auto simp add*: *h-def*)
have *h-outside-E*: *h n* $(x, y) = 0$ **if** $(x, y) \notin \mathbf{E}$ **for** *n x y* **using** *that* **unfolding** *h-def*
by(*clarsimp*)(*subst dom-h′*, *auto simp add*: *R-def*)

define *H* **where** *H xy* = *lim* ($\lambda n.$ *h* (*k n*) *xy*) **for** *xy*
have *H*: ($\lambda n.$ *h* (*k n*) *xy*) $\longrightarrow$ *H xy* **for** *xy*
using *conv*[*of xy*] **unfolding** *H-def* **by**(*cases xy* $\in$ *?Q*)(*auto simp add*: *convergent-LIMSEQ-iff h-outside*)
have *H-outside*: $H\ (x, y) = 0$ **if** $x \notin A\ \Gamma \vee y \notin B\ \Gamma$ **for** *x y*
using *that* **by**(*simp add*: *H-def convergent-LIMSEQ-iff h-outside-AB*)
have *H′*: ($\lambda n.$ *ennreal* (*h* (*k n*) *xy*)) $\longrightarrow$ *H xy* **for** *xy*
using *H* **by**(*rule tendsto-ennrealI*)
have *H-def′*: *H xy* = *lim* ($\lambda n.$ *ennreal* (*h* (*k n*) *xy*)) **for** *xy* **by** (*metis H′ limI*)

have *H-OUT*: *d-OUT H x* = *weight* Γ *x* **if** *x*: $x \in A\ \Gamma$ **for** *x*
proof −
let *?w* = $\lambda y.$ *if* $(x, y) \in \mathbf{E}$ *then weight* Γ *y else 0*
have *sum-w*: ($\sum^{+}$ *y. if edge* Γ *x y then weight* Γ *y else 0*) = ($\sum^{+}$ $y \in \mathbf{E}$ “ $\{x\}$. *weight* Γ *y*)
by(*simp add*: *nn-integral-count-space-indicator indicator-def of-bool-def if-distrib cong*: *if-cong*)
have ($\lambda n.$ *d-OUT* (*h* (*k n*)) *x*) $\longrightarrow$ *d-OUT H x* **unfolding** *d-OUT-def*
by(*rule nn-integral-dominated-convergence*[**where** *w*=*?w*])(*use bounded-B x* **in** ‹*simp-all add*: *AE-count-space H h-outside-E h-le-weight2 sum-w*›)
moreover define *n-x* **where** *n-x* = *to-nat-on* ($A\ \Gamma$) *x*
have *x′*: $x \in$ *A-n* (*k n*) **if** $n \geq$ *n-x* **for** *n*
using *that seq-suble*[*OF k*, *of n*] *x* **unfolding** *A-n-def*
by(*intro rev-image-eqI*[**where** *x*=*n-x*])(*simp-all add*: *A-n-def n-x-def*)
have ($\lambda n.$ *d-OUT* (*h* (*k n*)) *x*) $\longrightarrow$ *weight* Γ *x*
by(*intro tendsto-eventually eventually-sequentiallyI*[**where** *c*=*n-x*])(*simp add*: *h-OUT x′*)
ultimately show *?thesis* **by**(*rule LIMSEQ-unique*)
qed
then have *linkage* Γ *H* **..**
moreover
have *web-flow* Γ *H* **unfolding** *web-flow-iff*
proof
show *d-OUT H x* $\leq$ *weight* Γ *x* **for** *x*
by(*cases* $x \in A\ \Gamma$)(*simp-all add*: *H-OUT*[*unfolded d-OUT-def*] *H-outside d-OUT-def*)

show *d-IN H y* $\leq$ *weight* Γ *y* **for** *y*
proof −
have *d-IN H y* = ($\sum^{+}$ *x. liminf* ($\lambda n.$ *ennreal* (*h* (*k n*) (*x*, *y*)))) **unfolding** *d-IN-def H-def′*
by(*rule nn-integral-cong convergent-liminf-cl*[*symmetric*] *convergentI H′*)+
also have … $\leq$ *liminf* ($\lambda n.$ *d-IN* (*h* (*k n*)) *y*)

unfolding *d-IN-def* **by**(*rule nn-integral-liminf*) *simp*
also have ... $\leq$ *liminf* ($\lambda n.$ *weight* Γ *y*) **unfolding** *h-IN*
by(*rule Liminf-mono*)(*auto simp add*: *ennreal-enn2real-if*)
also have ... = *weight* Γ *y* **by**(*simp add*: *Liminf-const*)
finally show *?thesis* **.**
qed

show *ennreal* (*H e*) = *0* **if** $e \notin$ **E** **for** *e*
proof(*rule LIMSEQ-unique*[*OF H'*])
obtain *x y* **where** [*simp*]: *e* = (*x*, *y*) **by**(*cases e*)
have *ennreal* (*h* (*k n*) (*x*, *y*)) = *0* **for** *n*
using *dom-h'*[*of Some x y k n*] *that* **by**(*auto simp add*: *h-def R-def enn2real-eq-0-iff elim*: *meta-mp*)
then show ($\lambda n.$ *ennreal* (*h* (*k n*) *e*)) $\longrightarrow$ *0* **using** *that*
by(*intro tendsto-eventually eventually-sequentiallyI*) *simp*
qed
qed
ultimately show *?thesis* **by** *blast*
qed

end

8.2 Glueing the reductions together

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

lemma *bounded-countable-bipartite-web-of*: *bounded-countable-bipartite-web* (*bipartite-web-of* Γ)
(**is** *bounded-countable-bipartite-web* ?Γ)
proof −
interpret *bi*: *countable-bipartite-web* ?Γ **by**(*rule countable-bipartite-web-of*)
show *?thesis*
proof
fix *x*
assume $x \in A$?Γ
then obtain x' **where** *x*: $x = Inl\ x'$ **and** x': *vertex* Γ x' $x' \notin B\ \Gamma$ **by** *auto*
have $\mathbf{E}_{?\Gamma}$ `` $\{x\} \subseteq Inr$ ` ($\{x'\} \cup$ (**E** `` $\{x'\}$)) **using** *x*
by(*auto simp add*: *bipartite-web-of-def vertex-def split*: *sum.split-asm*)
hence $(\sum^+ y \in \mathbf{E}_{?\Gamma}$ `` $\{x\}.$ *weight* ?Γ *y*) $\leq (\sum^+ y \in \ldots.$ *weight* ?Γ *y*)
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... = $(\sum^+ y \in \{x'\} \cup$ (**E** `` $\{x'\}$). *weight* (*bipartite-web-of* Γ) (*Inr y*))
by(*rule nn-integral-count-space-reindex*)(*auto*)
also have ... $\leq$ *weight* Γ x' + $(\sum^+ y \in$ **E** `` $\{x'\}.$ *weight* Γ *y*)

apply(*subst* (*1 2*) *nn-integral-count-space-indicator*, *simp*, *simp*)
apply(*cases* $\neg$ *edge* Γ x' x')
apply(*subst nn-integral-disjoint-pair*)
apply(*auto intro*!: *nn-integral-mono add-increasing split*: *split-indicator*)
done
also have ... $< \top$ **using** *bounded-out*[*of* x'] x' **using** *weight-finite*[*of* x'] **by**(*simp del*: *weight-finite add*: *less-top*)
finally show $(\sum^+ y \in \mathbf{E}_{?\Gamma}$ '' $\{x\}$. *weight* $?\Gamma$ $y) < \top$ **.**
qed
qed

theorem *loose-linkable-bounded*:
assumes *loose* Γ
shows *linkable* Γ
proof $-$
interpret *bi*: *bounded-countable-bipartite-web bipartite-web-of* Γ **by**(*rule bounded-countable-bipartite-web-of*)
have $\neg$ *hindered* (*bipartite-web-of* Γ) **using** *assms* **by**(*rule unhindered-bipartite-web-of*)
then have *linkable* (*bipartite-web-of* Γ)
by(*rule bi.unhindered-linkable-bounded*)
then show *?thesis* **by**(*rule linkable-bipartite-web-ofD*) *simp*
qed

lemma *bounded-countable-web-quotient-web*: *bounded-countable-web* (*quotient-web* Γ f) (**is** *bounded-countable-web* $?\Gamma$)
proof $-$
interpret *r*: *countable-web* $?\Gamma$ **by**(*rule countable-web-quotient-web*)
show *?thesis*
proof
fix x
assume $x \in \mathbf{V}_{quotient\text{-}web\ \Gamma\ f} - B$ (*quotient-web* Γ f)
then have x: $x \in \mathbf{V} - B\ \Gamma$ **by**(*auto dest*: *vertex-quotient-webD*)
have $(\sum^+ y \in \mathbf{E}_{?\Gamma}$ '' $\{x\}$. *weight* $?\Gamma$ $y) \le (\sum^+ y \in \mathbf{E}$ '' $\{x\}$. *weight* Γ $y)$
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... $< \top$ **using** x **by**(*rule bounded-out*)
finally show $(\sum^+ y \in \mathbf{E}_{?\Gamma}$ '' $\{x\}$. *weight* $?\Gamma$ $y) < \top$ **.**
qed
qed

lemma *ex-orthogonal-current*:
$\exists f\ S$. *web-flow* Γ f $\wedge$ *separating* Γ S $\wedge$ *orthogonal-current* Γ f S
proof $-$
from *ex-maximal-wave*[*OF countable*]
obtain f **where** f: *current* Γ f
and w: *wave* Γ f
and *maximal*: $\bigwedge w$. $[\![$ *current* Γ w; *wave* Γ w; $f \le w$ $]\!] \Longrightarrow f = w$ **by** *blast*
from *ex-trimming*[*OF f w countable weight-finite*] **obtain** h **where** h: *trimming* Γ f h **..**

let *?Γ* = *quotient-web* Γ *f*
interpret Γ: *bounded-countable-web ?Γ* **by**(*rule bounded-countable-web-quotient-web*)
have *loose ?Γ* **using** *f w maximal* **by**(*rule loose-quotient-web*[*OF weight-finite*])
then have *linkable ?Γ* **by**(*rule* Γ.*loose-linkable-bounded*)
then obtain *g* **where** *wg*: *web-flow ?Γ g* **and** *link*: *linkage ?Γ g* **by** *blast*

let *?k* = *plus-current h g*
have *web-flow* Γ *?k orthogonal-current* Γ *?k* ($\mathcal{E}$ (*TER f*))
by(*rule linkage-quotient-webD*[*OF f w wg link h*])+
moreover have *separating* Γ ($\mathcal{E}$ (*TER f*))
using *waveD-separating*[*OF w*] **by**(*rule separating-essential*)
ultimately show *?thesis* **by** *blast*
qed

end

locale *bounded-countable-network* = *countable-network* Δ
for Δ :: (*'v*, *'more*) *network-scheme* (**structure**) +
assumes *out*: ⟦ $x \in$ **V**; $x \neq$ *source* Δ; $x \neq$ *sink* Δ ⟧ ⟹ *d-OUT* (*capacity* Δ) *x* < ⊤

context *antiparallel-edges* **begin**

lemma *Δ''-bounded-countable-network*: *bounded-countable-network* Δ''
if ⋀*x*. ⟦ $x \in$ **V**; $x \neq$ *source* Δ; $x \neq$ *sink* Δ ⟧ ⟹ *d-OUT* (*capacity* Δ) *x* < ⊤
proof −
interpret *ae*: *countable-network* Δ'' **by**(*rule Δ''-countable-network*)
show *?thesis*
proof
fix *x*
assume *x*: $x \in \mathbf{V}_{\Delta''}$ **and** *not-source*: $x \neq$ *source* Δ'' **and** *not-sink*: $x \neq$ *sink* Δ''
from *x* **consider** (*Vertex*) *x'* **where** *x* = *Vertex x' x'* ∈ **V** | (*Edge*) *y z* **where** *x* = *Edge y z edge* Δ *y z*
unfolding **V**-Δ'' **by** *auto*
then show *d-OUT* (*capacity* Δ'') *x* < ⊤
proof *cases*
case *Vertex*
then show *?thesis* **using** *x not-source not-sink that*[*of x'*] **by** *auto*
next
case *Edge*
then show *?thesis* **using** *capacity-finite*[*of* (*y*, *z*)] **by**(*simp del*: *capacity-finite add*: *less-top*)
qed
qed
qed

end

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

lemma *bounded-countable-web-web-of-network*:
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* $\Delta\ x\ x$
shows *bounded-countable-web* (*web-of-network* Δ) (**is** *bounded-countable-web* ?Γ)
proof $-$
interpret *web*: *countable-web* ?Γ **by**(*rule countable-web-web-of-network*) *fact+*
show *?thesis*
proof
fix *e*
assume $e \in \mathbf{V}_{?\Gamma} - B$?Γ
then obtain *x y* **where** *e*: $e = (x, y)$ **and** *xy*: *edge* $\Delta\ x\ y$ **by**(*cases e*) *simp*
from *xy* **have** *y*: $y \neq$ *source* Δ **using** *source-in*[*of x*] **by** *auto*
have *out-sink*: *d-OUT* (*capacity* Δ) (*sink* Δ) = *0* **unfolding** *d-OUT-def*
by(*auto simp add*: *nn-integral-0-iff-AE AE-count-space sink-out intro*!: *capacity-outside*)
have $\mathbf{E}_{?\Gamma}$ `` $\{e\} \subseteq \mathbf{E} \cap \{y\} \times$ *UNIV* **using** *e* **by** *auto*
hence $(\sum^+ e' \in \mathbf{E}_{?\Gamma}$ `` $\{e\}.$ *weight* ?Γ $e') \leq (\sum^+ e \in \mathbf{E} \cap \{y\} \times$ *UNIV*. *capacity* $\Delta\ e$) **using** *e*
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
also have $\ldots \leq (\sum^+ e \in$ *Pair y* ` *UNIV*. *capacity* $\Delta\ e$)
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
also have $\ldots =$ *d-OUT* (*capacity* Δ) *y* **unfolding** *d-OUT-def*
by(*rule nn-integral-count-space-reindex*) *simp*
also have $\ldots < \top$ **using** *out*[*of y*] *xy y out-sink* **by**(*cases* $y =$ *sink* Δ)(*auto simp add*: *vertex-def*)
finally show $(\sum^+ e' \in \mathbf{E}_{?\Gamma}$ `` $\{e\}.$ *weight* ?Γ $e') < \top$ **.**
qed
qed

context begin

qualified lemma *max-flow-min-cut′-bounded*:
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* $\Delta\ f\ S$
by(*rule max-flow-min-cut′*)(*rule bounded-countable-web.ex-orthogonal-current*[*OF bounded-countable-web-web-of-network*], *fact+*)

qualified lemma *max-flow-min-cut''-bounded*:
  assumes *sink-out*: $\bigwedge y.\ \neg$ *edge* Δ (*sink* Δ) *y*
  and *source-in*: $\bigwedge x.\ \neg$ *edge* Δ *x* (*source* 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 −
  interpret *antiparallel-edges* Δ **..**
  interpret Δ'': *bounded-countable-network* Δ'' **by**(*rule* Δ''*-bounded-countable-network*)(*rule out*)
  have $\exists f\ S.$ *flow* Δ'' *f* $\wedge$ *cut* Δ'' *S* $\wedge$ *orthogonal* Δ'' *f S*
   by(*rule* Δ''*.max-flow-min-cut'-bounded*)(*auto simp add*: *sink-out source-in no-loop capacity-pos elim*: *edg.cases*)
  then obtain *f S* **where** *f*: *flow* Δ'' *f* **and** *cut*: *cut* Δ'' *S* **and** *ortho*: *orthogonal* Δ'' *f S* **by** *blast*
  have *flow* Δ (*collect f*) **using** *f* **by**(*rule flow-collect*)
  moreover have *cut* Δ (*cut' S*) **using** *cut* **by**(*rule cut-cut'*)
  moreover have *orthogonal* Δ (*collect f*) (*cut' S*) **using** *ortho f* **by**(*rule orthogonal-cut'*)
  ultimately show *?thesis* **by** *blast*
qed

qualified lemma *max-flow-min-cut'''-bounded*:
  assumes *sink-out*: $\bigwedge y.\ \neg$ *edge* Δ (*sink* Δ) *y*
  and *source-in*: $\bigwedge x.\ \neg$ *edge* Δ *x* (*source* Δ)
  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 −
  interpret *antiparallel-edges* Δ **..**
  interpret Δ'': *bounded-countable-network* Δ'' **by**(*rule* Δ''*-bounded-countable-network*)(*rule out*)
  have $\exists f\ S.$ *flow* Δ'' *f* $\wedge$ *cut* Δ'' *S* $\wedge$ *orthogonal* Δ'' *f S*
    by(*rule* Δ''*.max-flow-min-cut''-bounded*)(*auto simp add*: *sink-out source-in capacity-pos elim*: *edg.cases*)
  then obtain *f S* **where** *f*: *flow* Δ'' *f* **and** *cut*: *cut* Δ'' *S* **and** *ortho*: *orthogonal* Δ'' *f S* **by** *blast*
  have *flow* Δ (*collect f*) **using** *f* **by**(*rule flow-collect*)
  moreover have *cut* Δ (*cut' S*) **using** *cut* **by**(*rule cut-cut'*)
  moreover have *orthogonal* Δ (*collect f*) (*cut' S*) **using** *ortho f* **by**(*rule orthogonal-cut'*)
  ultimately show *?thesis* **by** *blast*
qed

lemma Δ'''*-bounded-countable-network*: *bounded-countable-network* Δ'''
proof −
  interpret Δ''': *countable-network* Δ''' **by**(*rule* Δ'''*-countable-network*)
  show *?thesis*
  proof
    fix *x*

assume x: $x \in \mathbf{V}_{\Delta'''}$ **and** *not-source*: $x \neq$ *source* Δ''' **and** *not-sink*: $x \neq$ *sink* Δ'''
from x **have** x': $x \in \mathbf{V}$ **by**(*auto simp add*: *vertex-def*)
have *d-OUT* (*capacity* Δ''') $x \leq$ *d-OUT* (*capacity* Δ) x **by**(*rule d-OUT-mono*) *simp*
also have $\ldots < \top$ **using** x' *not-source not-sink* **by**(*simp add*: *out*)
finally show *d-OUT* (*capacity* Δ''') $x < \top$ **.**
qed
qed

theorem *max-flow-min-cut-bounded*:
$\exists f\, S$. *flow* Δ f $\wedge$ *cut* Δ S $\wedge$ *orthogonal* Δ f S
proof −
interpret Δ': *bounded-countable-network* Δ''' **by**(*rule* Δ'''*-bounded-countable-network*)
have $\exists f\, S$. *flow* Δ''' f $\wedge$ *cut* Δ''' S $\wedge$ *orthogonal* Δ''' f S **by**(*rule* Δ'*.max-flow-min-cut*$'''$*-bounded*) *auto*
then obtain $f\, S$ **where** f: *flow* Δ''' f **and** *cut*: *cut* Δ''' S **and** *ortho*: *orthogonal* Δ''' f S **by** *blast*
from *flow-*Δ'''[*OF this*] **show** *?thesis* **by** *blast*
qed

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)
by(*simp add*: *cleanup-def*)

lemma *value-flow-cleanup*:
assumes [*simp*]: $\bigwedge x.\ f\ (x,\ source\ \Delta) = 0$
shows *value-flow* Δ (*cleanup* f) $=$ *value-flow* Δ f
unfolding *d-OUT-def*
by (*auto simp add*: *not-less intro*!: *nn-integral-cong intro*: *antisym*)

lemma *KIR-cleanup*:
assumes *KIR*: *KIR f x*
and *finite-IN*: *d-IN f x* $\neq \top$
shows *KIR* (*cleanup f*) *x*
proof $-$
from *finite-IN KIR* **have** *finite-OUT*: *d-OUT f x* $\neq \top$ **by** *simp*

have *finite-IN*: $(\sum^+ y{\in}A.\ f\ (y,\ x)) \neq \top$ **for** *A*
using *finite-IN* **by**(*rule neq-top-trans*)(*auto simp add*: *d-IN-def nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
have *finite-OUT*: $(\sum^+ y{\in}A.\ f\ (x,\ y)) \neq \top$ **for** *A*
using *finite-OUT* **by**(*rule neq-top-trans*)(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
have *finite-in*: $f\ (x,\ y) \neq \top$ **for** *y* **using** ‹*d-OUT f x* $\neq \top$›
by(*rule neq-top-trans*) (*rule d-OUT-ge-point*)

let $?M = \{y.\ f\ (x,\ y) > f\ (y,\ x)\}$

have *d-OUT* (*cleanup f*) $x = (\sum^+ y{\in}?M.\ f\ (x,\ y) - f\ (y,\ x))$
by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator intro*!: *nn-integral-cong*)
also have $\ldots = (\sum^+ y{\in}?M.\ f\ (x,\ y)) - (\sum^+ y{\in}?M.\ f\ (y,\ x))$ **using** *finite-IN*
by(*subst nn-integral-diff*)(*auto simp add*: *AE-count-space*)
also have $\ldots = ($*d-OUT f x* $- (\sum^+ y{\in}\{y.\ f\ (x,\ y) \leq f\ (y,\ x)\}.\ f\ (x,\ y))) - (\sum^+ y{\in}?M.\ f\ (y,\ x))$
unfolding *d-OUT-def d-IN-def* **using** *finite-IN finite-OUT*
apply(*simp add*: *nn-integral-count-space-indicator*)
apply(*subst* (*2*) *nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space finite-in split*: *split-indicator intro*!: *arg-cong2*[**where** $f=(-)$] *intro*!: *nn-integral-cong*)
done
also have $\ldots = ($*d-IN f x* $- (\sum^+ y{\in}?M.\ f\ (y,\ x))) - (\sum^+ y{\in}\{y.\ f\ (x,\ y) \leq f\ (y,\ x)\}.\ f\ (x,\ y))$
using *KIR* **by**(*simp add*: *diff-diff-commute-ennreal*)
also have $\ldots = (\sum^+ y{\in}\{y.\ f\ (x,\ y) \leq f\ (y,\ x)\}.\ f\ (y,\ x)) - (\sum^+ y{\in}\{y.\ f\ (x,\ y) \leq f\ (y,\ x)\}.\ f\ (x,\ y))$
using *finite-IN finite-IN*[*of* { - }]
apply(*simp add*: *d-IN-def nn-integral-count-space-indicator*)
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *d-IN-def AE-count-space split*: *split-indicator intro*!: *arg-cong2*[**where** $f=(-)$] *intro*!: *nn-integral-cong*)
done
also have $\ldots = (\sum^+ y{\in}\{y.\ f\ (x,\ y) \leq f\ (y,\ x)\}.\ f\ (y,\ x) - f\ (x,\ y))$ **using** *finite-OUT*
by(*subst nn-integral-diff*)(*auto simp add*: *AE-count-space*)
also have $\ldots =$ *d-IN* (*cleanup f*) *x* **using** *finite-in*
by(*auto simp add*: *d-IN-def nn-integral-count-space-indicator intro*!: *ennreal-diff-self nn-integral-cong split*: *split-indicator*)
finally show *KIR* (*cleanup f*) *x* **.**
qed

locale *flow-attainability* = *countable-network* Δ
  for Δ :: (′*v*, ′*more*) *network-scheme* (**structure**)
  +
  assumes *finite-capacity*: ⋀*x*. *x* ≠ *sink* Δ ⟹ *d-IN* (*capacity* Δ) *x* ≠ ⊤ ∨ *d-OUT* (*capacity* Δ) *x* ≠ ⊤
  and *no-loop*: ⋀*x*. ¬ *edge* Δ *x x*
  and *source-in*: ⋀*x*. ¬ *edge* Δ *x* (*source* Δ)
begin

lemma *source-in-not-cycle*:
  assumes *cycle* Δ *p*
  shows (*x*, *source* Δ) ∉ *set* (*cycle-edges p*)
using *cycle-edges-edges*[*OF assms*] *source-in*[*of x*] **by**(*auto*)

lemma *source-out-not-cycle*:
  cycle Δ *p* ⟹ (*source* Δ, *x*) ∉ *set* (*cycle-edges p*)
by(*auto dest*: *cycle-leave-ex-enter source-in-not-cycle*)

lemma *flowD-source-IN*:
  assumes *flow* Δ *f*
  shows *d-IN f* (*source* Δ) = *0*
proof −
  have *d-IN f* (*source* Δ) = ($\sum^+$ *y*∈**IN** (*source* Δ). *f* (*y*, *source* Δ))
    by(*rule d-IN-alt-def*)(*simp add*: *flowD-outside*[*OF assms*])
  also have ... = ($\sum^+$ *y*∈**IN** (*source* Δ). *0*)
    by(*rule nn-integral-cong*)(*simp add*: *source-in incoming-def*)
  finally show *?thesis* **by** *simp*
qed

lemma *flowD-finite-IN*:
  assumes *f*: *flow* Δ *f* **and** *x*: *x* ≠ *sink* Δ
  shows *d-IN f x* ≠ *top*
proof(*cases x* = *source* Δ)
  case *True* **thus** *?thesis* **by**(*simp add*: *flowD-source-IN*[*OF f*])
next
  case *False*
  from *finite-capacity*[*OF x*] **show** *?thesis*
  proof
    assume ∗: *d-IN* (*capacity* Δ) *x* ≠ ⊤
     from *flowD-capacity*[*OF f*] **have** *d-IN f x* ≤ *d-IN* (*capacity* Δ) *x* **by**(*rule d-IN-mono*)
    also have ... < ⊤ **using** ∗ **by** (*simp add*: *less-top*)
    finally show *?thesis* **by** *simp*
  next
    assume ∗: *d-OUT* (*capacity* Δ) *x* ≠ ⊤
    have *d-IN f x* = *d-OUT f x* **using** *flowD-KIR*[*OF f False x*] **by** *simp*
     also have ... ≤ *d-OUT* (*capacity* Δ) *x* **using** *flowD-capacity*[*OF f*] **by**(*rule d-OUT-mono*)

also have ... $< \top$ **using** $*$ **by** (*simp add*: *less-top*)
finally show *?thesis* **by** *simp*
qed
qed

lemma *flowD-finite-OUT*:
assumes *flow* Δ *f x* $\neq$ *source* Δ *x* $\neq$ *sink* Δ
shows *d-OUT f x* $\neq \top$
using *flowD-KIR*[*OF assms*] *assms* **by**(*simp add*: *flowD-finite-IN*)

end

locale *flow-network* = *flow-attainability*
+
fixes *g* :: *'v flow*
assumes *g*: *flow* Δ *g*
and *g-finite*: *value-flow* Δ *g* $\neq \top$
and *nontrivial*: $\mathbf{V}$ − {*source* Δ, *sink* Δ} $\neq$ {}
begin

lemma *g-outside*: *e* $\notin$ $\mathbf{E}$ $\Longrightarrow$ *g e* = *0*
by(*rule flowD-outside*)(*rule g*)

lemma *g-loop* [*simp*]: *g* (*x*, *x*) = *0*
by(*rule g-outside*)(*simp add*: *no-loop*)

lemma *finite-IN-g*: *x* $\neq$ *sink* Δ $\Longrightarrow$ *d-IN g x* $\neq$ *top*
by(*rule flowD-finite-IN*[*OF g*])

lemma *finite-OUT-g*:
assumes *x* $\neq$ *sink* Δ
shows *d-OUT g x* $\neq$ *top*
proof(*cases x* = *source* Δ)
case *True*
with *g-finite* **show** *?thesis* **by** *simp*
next
case *False*
with *g* **have** *KIR g x* **using** *assms* **by**(*auto dest*: *flowD-KIR*)
with *finite-IN-g*[*of x*] *False assms* **show** *?thesis* **by**(*simp*)
qed

lemma *g-source-in* [*simp*]: *g* (*x*, *source* Δ) = *0*
by(*rule g-outside*)(*simp add*: *source-in*)

lemma *finite-g* [*simp*]: *g e* $\neq$ *top*
by(*rule flowD-finite*[*OF g*])

definition *enum-v* :: *nat* $\Rightarrow$ *'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}
using from-nat-into[OF nontrivial] by(auto simp add: enum-v-def)

lemma enum-v-repeat:
  assumes x: x ∈ V x ≠ source Δ x ≠ sink Δ
  shows ∃ i'>i. enum-v i' = x
proof −
  let ?V = V − {source Δ, sink Δ}
  let ?n = to-nat-on ?V x
  let ?A = {?n} × (UNIV :: nat set)
  from x have x': x ∈ V − {source Δ, sink Δ} by simp
  have infinite ?A by(auto dest: finite-cartesian-productD2)
 hence infinite (prod-encode ‘ ?A) by(auto dest: finite-imageD simp add: inj-prod-encode)
  then obtain i' where i' > i i' ∈ prod-encode ‘ ?A
    unfolding infinite-nat-iff-unbounded by blast
  from this(2) have enum-v i' = x using x by(clarsimp simp add: enum-v-def)
  with ‹i' > i› show ?thesis by blast
qed

fun h-plus :: nat ⇒ 'v edge ⇒ ennreal
where
  h-plus 0 (x, y) = (if x = source Δ then g (x, y) else 0)
| h-plus (Suc i) (x, y) =
  (if enum-v (Suc i) = x ∧ d-OUT (h-plus i) x < d-IN (h-plus i) x then
   let total = d-IN (h-plus i) x − d-OUT (h-plus i) x;
       share = g (x, y) − h-plus i (x, y);
       shares = d-OUT g x − d-OUT (h-plus i) x
   in h-plus i (x, y) + share ∗ total / shares
   else h-plus i (x, y))


lemma h-plus-le-g: h-plus i e ≤ g e
proof(induction i arbitrary: e and e)
  case 0 thus ?case by(cases e) simp
next
  case (Suc i)
  { fix x y
    assume enum: x = enum-v (Suc i)
    assume less: d-OUT (h-plus i) x < d-IN (h-plus i) x
    from enum have x: x ≠ source Δ x ≠ sink Δ using range-enum-v
      by(auto dest: sym intro: rev-image-eqI)

    define share where share = g (x, y) − h-plus i (x, y)
    define shares where shares = d-OUT g x − d-OUT (h-plus i) x
    define total where total = d-IN (h-plus i) x − d-OUT (h-plus i) x
    let ?h = h-plus i (x, y) + share ∗ total / shares
```

have *d-OUT* (*h-plus i*) $x \leq$ *d-OUT g x* **by**(*rule d-OUT-mono*)(*rule Suc.IH*)
also have ... $<$ *top* **using** *finite-OUT-g*[*of x*] *x* **by** (*simp add*: *less-top*)
finally have *d-OUT* (*h-plus i*) $x \neq \top$ **by** *simp*
then have *shares-eq*: *shares* = ($\sum^+$ *y*. *g* (*x*, *y*) − *h-plus i* (*x*, *y*)) **unfolding** *shares-def d-OUT-def*
by(*subst nn-integral-diff*)(*simp-all add*: *AE-count-space Suc.IH*)

have ∗: *share* / *shares* ≤ *1*
proof (*cases share* = *0*)
case *True* **thus** *?thesis* **by**(*simp*)
next
case *False*
hence *share* > *0* **using** ‹*h-plus i* (*x*, *y*) ≤ *g* -›
by(*simp add*: *share-def dual-order.strict-iff-order*)
moreover have *share* ≤ *shares* **unfolding** *share-def shares-eq* **by**(*rule nn-integral-ge-point*)*simp*
ultimately show *?thesis* **by**(*simp add*: *divide-le-posI-ennreal*)
qed

note *shares-def*
also have *d-OUT g x* = *d-IN g x* **by**(*rule flowD-KIR*[*OF g x*])
also have *d-IN* (*h-plus i*) *x* ≤ *d-IN g x* **by**(*rule d-IN-mono*)(*rule Suc.IH*)
ultimately have ∗: *total* ≤ *shares* **unfolding** *total-def* **by**(*simp add*: *ennreal-minus-mono*)
moreover have *total* > *0* **unfolding** *total-def* **using** *less* **by** (*clarsimp simp add*: *diff-gr0-ennreal*)
ultimately have *total* / *shares* ≤ *1* **by**(*intro divide-le-posI-ennreal*)(*simp-all*)
hence *share* ∗ (*total* / *shares*) ≤ *share* ∗ *1*
by(*rule mult-left-mono*) *simp*
hence *?h* ≤ *h-plus i* (*x*, *y*) + *share* **by**(*simp add*: *ennreal-times-divide add-mono*)
also have ... = *g* (*x*, *y*) **unfolding** *share-def* **using** ‹*h-plus i* (*x*, *y*) ≤ *g* -› *finite-g*[*of* (*x*, *y*)]
by *simp*
moreover
note *calculation* **}**
note ∗ = *this*
show *?case* **using** *Suc.IH* ∗ **by**(*cases e*) *clarsimp*
qed

lemma *h-plus-outside*: $e \notin \mathbf{E} \Longrightarrow$ *h-plus i e* = *0*
by (*metis g-outside h-plus-le-g le-zero-eq*)

lemma *h-plus-not-infty* [*simp*]: *h-plus i e* ≠ *top*
using *h-plus-le-g*[*of i e*] **by** (*auto simp*: *top-unique*)

lemma *h-plus-mono*: *h-plus i e* ≤ *h-plus* (*Suc i*) *e*
proof(*cases e*)
case [*simp*]: (*Pair x y*)
{ assume *d-OUT* (*h-plus i*) *x* < *d-IN* (*h-plus i*) *x*

hence *h-plus i* $(x, y) + 0 \leq$ *h-plus i* $(x, y) + (g\ (x, y) -$ *h-plus i* $(x, y)) *$ (*d-IN* (*h-plus i*) $x -$ *d-OUT* (*h-plus i*) x) / (*d-OUT g x* $-$ *d-OUT* (*h-plus i*) x)
by(*intro add-left-mono d-OUT-mono le-funI*) (*simp-all add*: *h-plus-le-g*) **}**
then show *?thesis* **by** *clarsimp*
qed

lemma *h-plus-mono′*: $i \leq j \Longrightarrow$ *h-plus i e* $\leq$ *h-plus j e*
by(*induction rule*: *dec-induct*)(*auto intro*: *h-plus-mono order-trans*)

lemma *d-OUT-h-plus-not-infty′*: $x \neq$ *sink* $\Delta \Longrightarrow$ *d-OUT* (*h-plus i*) $x \neq$ *top*
using *d-OUT-mono*[*of h-plus i x g*, *OF h-plus-le-g*] *finite-OUT-g*[*of x*] **by** (*auto simp*: *top-unique*)

lemma *h-plus-OUT-le-IN*:
assumes $x \neq$ *source* Δ
shows *d-OUT* (*h-plus i*) $x \leq$ *d-IN* (*h-plus i*) x
proof(*induction i*)
case *0*
thus *?case* **using** *assms* **by**(*simp add*: *d-OUT-def*)
next
case (*Suc i*)
have *d-OUT* (*h-plus* (*Suc i*)) $x \leq$ *d-IN* (*h-plus i*) x
proof(*cases enum-v* (*Suc i*) $= x \wedge$ *d-OUT* (*h-plus i*) $x <$ *d-IN* (*h-plus i*) x)
case *False*
thus *?thesis* **using** *Suc.IH* **by**(*simp add*: *d-OUT-def cong*: *conj-cong*)
next
case *True*
hence *x*: $x \neq$ *sink* Δ **and** *le*: *d-OUT* (*h-plus i*) $x <$ *d-IN* (*h-plus i*) x **using** *range-enum-v* **by** *auto*
let *?r* $= \lambda y.$ $(g\ (x, y) -$ *h-plus i* $(x, y)) *$ (*d-IN* (*h-plus i*) $x -$ *d-OUT* (*h-plus i*) x) / (*d-OUT g x* $-$ *d-OUT* (*h-plus i*) x)
have *d-OUT* (*h-plus* (*Suc i*)) $x =$ *d-OUT* (*h-plus i*) $x + (\sum^+ y.$ *?r y*)
using *True* **unfolding** *d-OUT-def h-plus.simps* **by**(*simp add*: *AE-count-space nn-integral-add*)
also from *True* **have** $x \neq$ *source* Δ $x \neq$ *sink* Δ **using** *range-enum-v* **by** *auto*
from *flowD-KIR*[*OF g this*] *le d-IN-mono*[*of h-plus i x g*, *OF h-plus-le-g*]
have *le′*: *d-OUT* (*h-plus i*) $x <$ *d-OUT g x* **by**(*simp*)
then have $(\sum^+ y.$ *?r y*) $=$
(*d-IN* (*h-plus i*) $x -$ *d-OUT* (*h-plus i*) x) $* ((\sum^+ y.\ g\ (x, y) -$ *h-plus i* $(x, y))$ / (*d-OUT g x* $-$ *d-OUT* (*h-plus i*) x))
by(*subst mult.commute*, *subst ennreal-times-divide*[*symmetric*])
(*simp add*: *nn-integral-cmult nn-integral-divide Suc.IH diff-gr0-ennreal*)
also have $(\sum^+ y.\ g\ (x, y) -$ *h-plus i* $(x, y)) =$ *d-OUT g x* $-$ *d-OUT* (*h-plus i*) x **using** *x*
by(*subst nn-integral-diff*)(*simp-all add*: *d-OUT-def*[*symmetric*] *h-plus-le-g d-OUT-h-plus-not-infty′*)
also have ... / ... $= 1$ **using** *le′ finite-OUT-g*[*of x*] *x*
by(*auto intro*!: *ennreal-divide-self dest*: *diff-gr0-ennreal simp*: *less-top*[*symmetric*])
also have *d-OUT* (*h-plus i*) $x +$ (*d-IN* (*h-plus i*) $x -$ *d-OUT* (*h-plus i*) x) $*$

1 = *d-IN* (*h-plus* *i*) *x* **using** *x*
by (*simp add*: *Suc*)
finally show *?thesis* **by** *simp*
qed
also have ... $\leq$ *d-IN* (*h-plus* (*Suc* *i*)) *x* **by**(*rule d-IN-mono*)(*rule h-plus-mono*)
finally show *?case* **.**
qed

lemma *h-plus-OUT-eq-IN*:
assumes *enum*: *enum-v* (*Suc* *i*) = *x*
shows *d-OUT* (*h-plus* (*Suc* *i*)) *x* = *d-IN* (*h-plus* *i*) *x*
proof(*cases d-OUT* (*h-plus* *i*) *x* < *d-IN* (*h-plus* *i*) *x*)
case *False*
from *enum* **have** *x* $\neq$ *source* Δ **using** *range-enum-v* **by** *auto*
from *h-plus-OUT-le-IN*[*OF this*, *of i*] *False* **have** *d-OUT* (*h-plus* *i*) *x* = *d-IN* (*h-plus* *i*) *x* **by** *auto*
with *False enum* **show** *?thesis* **by**(*simp add*: *d-OUT-def*)
next
case *True*
from *enum* **have** *x*: *x* $\neq$ *source* Δ **and** *sink*: *x* $\neq$ *sink* Δ **using** *range-enum-v* **by** *auto*
let *?r* = λ*y*. (*g* (*x*, *y*) − *h-plus* *i* (*x*, *y*)) ∗ (*d-IN* (*h-plus* *i*) *x* − *d-OUT* (*h-plus* *i*) *x*) / (*d-OUT* *g* *x* − *d-OUT* (*h-plus* *i*) *x*)
have *d-OUT* (*h-plus* (*Suc* *i*)) *x* = *d-OUT* (*h-plus* *i*) *x* + ($\sum^+$ *y*. *?r y*)
using *True enum* **unfolding** *d-OUT-def h-plus.simps* **by**(*simp add*: *AE-count-space nn-integral-add*)
also from *True enum* **have** *x* $\neq$ *source* Δ *x* $\neq$ *sink* Δ **using** *range-enum-v* **by** *auto*
from *flowD-KIR*[*OF g this*] *True d-IN-mono*[*of h-plus i x g*, *OF h-plus-le-g*]
have *le'*: *d-OUT* (*h-plus* *i*) *x* < *d-OUT* *g* *x* **by**(*simp*)
then have ($\sum^+$ *y*. *?r y*) =
(*d-IN* (*h-plus* *i*) *x* − *d-OUT* (*h-plus* *i*) *x*) ∗ (($\sum^+$ *y*. *g* (*x*, *y*) − *h-plus* *i* (*x*, *y*)) / (*d-OUT* *g* *x* − *d-OUT* (*h-plus* *i*) *x*))
by(*subst mult.commute*, *subst ennreal-times-divide*[*symmetric*])
(*simp add*: *nn-integral-cmult nn-integral-divide h-plus-OUT-le-IN*[*OF x*] *diff-gr0-ennreal*)
also have ($\sum^+$ *y*. *g* (*x*, *y*) − *h-plus* *i* (*x*, *y*)) = *d-OUT* *g* *x* − *d-OUT* (*h-plus* *i*) *x* **using** *sink*
by(*subst nn-integral-diff*)(*simp-all add*: *d-OUT-def*[*symmetric*] *h-plus-le-g d-OUT-h-plus-not-infty'*)
also have ... / ... = *1* **using** *le' finite-OUT-g*[*of x*] *sink*
by(*auto intro*!: *ennreal-divide-self dest*: *diff-gr0-ennreal simp*: *less-top*[*symmetric*])
also have *d-OUT* (*h-plus* *i*) *x* + (*d-IN* (*h-plus* *i*) *x* − *d-OUT* (*h-plus* *i*) *x*) ∗ *1* = *d-IN* (*h-plus* *i*) *x* **using** *sink*
by (*simp add*: *h-plus-OUT-le-IN x*)
finally show *?thesis* **.**
qed

lemma *h-plus-source-in* [*simp*]: *h-plus* *i* (*x*, *source* Δ) = *0*
by(*induction i*)*simp-all*

lemma *h-plus-sum-finite*: $(\sum^+ e.\ h\text{-}plus\ i\ e) \neq top$
proof(*induction i*)
  case *0*
  have $(\sum^+ e{\in}UNIV.\ h\text{-}plus\ 0\ e) = (\sum^+ (x,\ y).\ h\text{-}plus\ 0\ (x,\ y))$
    by(*simp del*: *h-plus.simps*)
  also have $\ldots = (\sum^+ (x,\ y){\in}range\ (Pair\ (source\ \Delta)).\ h\text{-}plus\ 0\ (x,\ y))$
    by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-cong*)
  also have $\ldots$ = *value-flow* Δ *g* **by**(*simp add*: *d-OUT-def nn-integral-count-space-reindex*)
  also have $\ldots < \top$ **using** *g-finite* **by** (*simp add*: *less-top*)
  finally show *?case* **by** *simp*
next
  case (*Suc i*)
  define *xi* **where** *xi* = *enum-v* (*Suc i*)
  then have *xi*: *xi* $\neq$ *source* Δ *xi* $\neq$ *sink* Δ **using** *range-enum-v* **by** *auto*
  show *?case*
  proof(*cases d-OUT* (*h-plus i*) *xi* < *d-IN* (*h-plus i*) *xi*)
    case *False*
    hence $(\sum^+ e{\in}UNIV.\ h\text{-}plus\ (Suc\ i)\ e) = (\sum^+ e.\ h\text{-}plus\ i\ e)$
      by(*auto intro*!: *nn-integral-cong simp add*: *xi-def*)
    with *Suc.IH* **show** *?thesis* **by** *simp*
  next
    case *True*
    have *less*: *d-OUT* (*h-plus i*) *xi* < *d-OUT g xi*
      using *True flowD-KIR*[*OF g xi*] *d-IN-mono*[*of h-plus i xi*, *OF h-plus-le-g*]
      by *simp*

    have $(\sum^+ e.\ h\text{-}plus\ (Suc\ i)\ e) =$
      $(\sum^+ e{\in}UNIV.\ h\text{-}plus\ i\ e) + (\sum^+ (x,\ y).\ ((g\ (x,\ y) - h\text{-}plus\ i\ (x,\ y)) *$
(*d-IN* (*h-plus i*) *x* − *d-OUT* (*h-plus i*) *x*) / (*d-OUT g x* − *d-OUT* (*h-plus i*) *x*))
∗ *indicator* (*range* (*Pair xi*)) (*x*, *y*))
      (**is** - = *?IH* + *?rest* **is** - = - + $\int^+ (x,\ y).\ ?f\ x\ y * \text{-}\ \partial\text{-})$ **using** *xi True*
    by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *xi-def split-beta AE-count-space intro*!: *nn-integral-cong split*: *split-indicator intro*!: *h-plus-le-g h-plus-OUT-le-IN d-OUT-mono le-funI*)
    also have $?rest = (\sum^+ (x,\ y){\in}range\ (Pair\ xi).\ ?f\ x\ y)$
      by(*simp add*: *nn-integral-count-space-indicator split-def*)
    also have $\ldots = (\sum^+ y.\ ?f\ xi\ y)$ **by**(*simp add*: *nn-integral-count-space-reindex*)
    also have $\ldots = (\sum^+ y.\ g\ (xi,\ y) - h\text{-}plus\ i\ (xi,\ y)) * ((d\text{-}IN\ (h\text{-}plus\ i)\ xi -$
d-OUT (*h-plus i*) *xi*) / (*d-OUT g xi* − *d-OUT* (*h-plus i*) *xi*))
      (**is** - = *?integral* ∗ *?factor*) **using** *True less*
    by(*simp add*: *nn-integral-multc nn-integral-divide diff-gr0-ennreal ennreal-times-divide*)
    also have *?integral* = *d-OUT g xi* − *d-OUT* (*h-plus i*) *xi* **unfolding** *d-OUT-def* **using** *xi*
      by(*subst nn-integral-diff*)(*simp-all add*: *h-plus-le-g d-OUT-def*[*symmetric*] *d-OUT-h-plus-not-infty'*)
    also have $\ldots * ?factor$ = (*d-IN* (*h-plus i*) *xi* − *d-OUT* (*h-plus i*) *xi*) **using** *xi*
      apply (*subst ennreal-times-divide*)
      apply (*subst mult.commute*)
      apply (*subst ennreal-mult-divide-eq*)

apply (*simp-all add*: *diff-gr0-ennreal finite-OUT-g less zero-less-iff-neq-zero*[*symmetric*])
done
also have ... $\neq \top$ **using** *h-plus-OUT-eq-IN*[*OF refl*, *of i*, *folded xi-def*, *symmetric*] *xi*
by(*simp add*: *d-OUT-h-plus-not-infty'*)
ultimately show *?thesis* **using** *Suc.IH* **by** *simp*
qed
qed

lemma *d-OUT-h-plus-not-infty* [*simp*]: *d-OUT* (*h-plus i*) $x \neq$ *top*
proof −
have *d-OUT* (*h-plus i*) $x \leq (\sum^{+} y{\in}UNIV.\ \sum^{+} x.$ *h-plus i* $(x, y))$
unfolding *d-OUT-def* **by**(*rule nn-integral-mono nn-integral-ge-point*)+ *simp*
also have ... $< \top$ **using** *h-plus-sum-finite* **by**(*simp add*: *nn-integral-snd-count-space less-top*)
finally show *?thesis* **by** *simp*
qed

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

lemma *cycle-enum-cycle* [*simp*]: *cycles* $\Delta \neq \{\} \Longrightarrow$ *cycle* Δ (*enum-cycle n*)
unfolding *enum-cycle-def* **using** *from-nat-into*[*of cycles* Δ *n*] **by** *simp*

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

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

lemma *h-minus-aux-le-h'*: *h-minus-aux j e* $\leq$ *h' e*
proof(*induction j e rule*: *h-minus-aux.induct*)
case *0*: (*1 e*) **show** *?case* **by** *simp*
next
case *Suc*: (*2 j e*)
{ assume *e*: *e* $\in$ *set* (*cycle-edges* (*enum-cycle j*))
then have *h-minus-aux j e* + *Min* $\{$*h' e'* − *h-minus-aux j e'* $|$*e'*. *e'* $\in$ *set* (*cycle-edges* (*enum-cycle j*))$\}$ $\leq$
h-minus-aux j e + (*h' e* − *h-minus-aux j e*)
using [[*simproc add*: *finite-Collect*]] **by**(*cases e rule*: *prod.exhaust*)(*auto intro*!: *add-mono Min-le*)

also have ... $=$ *h′ e* **using** *e finite-h′*[*of e*] *Suc.IH*(*2*)[*of e*]
by(*cases e rule*: *prod.exhaust*)
(*auto simp add*: *add-diff-eq-ennreal top-unique intro*!: *ennreal-add-diff-cancel-left*)
also note *calculation* **}**
then show *?case* **using** *Suc* **by** *clarsimp*
qed

lemma *h-minus-aux-finite* [*simp*]: *h-minus-aux j e* $\neq$ *top*
using *h-minus-aux-le-h′*[*of j e*] *finite-h′*[*of e*] **by** (*auto simp*: *top-unique*)

lemma *h-minus-aux-mono*: *h-minus-aux j e* $\leq$ *h-minus-aux* (*Suc j*) *e*
proof(*cases e* $\in$ *set* (*cycle-edges* (*enum-cycle j*)) $=$ *True*)
case *True*
have *h-minus-aux j e* $+$ *0* $\leq$ *h-minus-aux* (*Suc j*) *e* **unfolding** *h-minus-aux.simps True if-True*
using *True* [[*simproc add*: *finite-Collect*]]
by(*cases e*)(*rule add-mono*, *auto intro*!: *Min.boundedI simp add*: *h-minus-aux-le-h′*)
thus *?thesis* **by** *simp*
qed *simp*

lemma *d-OUT-h-minus-aux*:
assumes *cycles* $\Delta \neq \{\}$
shows *d-OUT* (*h-minus-aux j*) *x* $=$ *d-IN* (*h-minus-aux j*) *x*
proof(*induction j*)
case *0* **show** *?case* **by** *simp*
next
case (*Suc j*)
define *C* **where** *C* $=$ *enum-cycle j*
define δ **where** δ $=$ *Min* {*h′ e′* $-$ *h-minus-aux j e′* |*e′*. *e′* $\in$ *set* (*cycle-edges C*)}

have *d-OUT* (*h-minus-aux* (*Suc j*)) *x* $=$
($\sum^+$ *y*. *h-minus-aux j* (*x*, *y*) $+$ (*if* (*x*, *y*) $\in$ *set* (*cycle-edges C*) *then* δ *else 0*))
unfolding *d-OUT-def* **by**(*simp add*: *if-distrib C-def* δ*-def cong del*: *if-weak-cong*)
also have ... $=$ *d-OUT* (*h-minus-aux j*) *x* $+$ ($\sum^+$ *y*. δ $*$ *indicator* (*set* (*cycle-edges C*)) (*x*, *y*))
(**is** - $=$ - $+$ *?add*)
by(*subst nn-integral-add*)(*auto simp add*: *AE-count-space d-OUT-def intro*!: *arg-cong2*[**where** *f*=(+)] *nn-integral-cong*)
also have *?add* $=$ ($\sum^+$ *e*$\in$*range* (*Pair x*). δ $*$ *indicator* {(*x′*, *y*). (*x′*, *y*) $\in$ *set* (*cycle-edges C*) $\wedge$ *x′* $=$ *x*} *e*)
by(*auto simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... $=$ δ $*$ *card* (*set* (*filter* (λ(*x′*, *y*). *x′* $=$ *x*) (*cycle-edges C*)))
using [[*simproc add*: *finite-Collect*]]
apply(*subst nn-integral-cmult-indicator*; *auto*)
apply(*subst emeasure-count-space*; *auto simp add*: *split-def*)
done
also have *card* (*set* (*filter* (λ(*x′*, *y*). *x′* $=$ *x*) (*cycle-edges C*))) $=$ *card* (*set* (*filter* (λ(*x′*, *y*). *y* $=$ *x*) (*cycle-edges C*)))

unfolding *C-def* **by**(*rule cycle-enter-leave-same*)(*rule cycle-enum-cycle*[*OF assms*])
also have $\delta * \ldots = (\sum^+ e{\in}range\ (\lambda x'.\ (x',\ x)).\ \delta * indicator\ \{(x',\ y).\ (x',\ y) \in set\ (cycle\text{-}edges\ C) \wedge y = x\}\ e)$
using [[*simproc add*: *finite-Collect*]]
apply(*subst nn-integral-cmult-indicator*; *auto*)
apply(*subst emeasure-count-space*; *auto simp add*: *split-def*)
done
also have $\ldots = (\sum^+ x'.\ \delta * indicator\ (set\ (cycle\text{-}edges\ C))\ (x',\ x))$
by(*auto simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-cong split*: *split-indicator*)
also have $d\text{-}OUT\ (h\text{-}minus\text{-}aux\ j)\ x + \ldots = (\sum^+ x'.\ h\text{-}minus\text{-}aux\ j\ (x',\ x) + \delta * indicator\ (set\ (cycle\text{-}edges\ C))\ (x',\ x))$
unfolding *Suc.IH d-IN-def* **by**(*simp add*: *nn-integral-add*[*symmetric*])
also have $\ldots = d\text{-}IN\ (h\text{-}minus\text{-}aux\ (Suc\ j))\ x$ **unfolding** *d-IN-def*
by(*auto intro*!: *nn-integral-cong simp add*: *δ-def C-def split*: *split-indicator*)
finally show *?case* **.**
qed

lemma *h-minus-aux-source*:
assumes $cycles\ \Delta \neq \{\}$
shows $h\text{-}minus\text{-}aux\ j\ (source\ \Delta,\ y) = 0$
proof(*induction j*)
case *0* **thus** *?case* **by** *simp*
next
case (*Suc j*)
have $(source\ \Delta,\ y) \notin set\ (cycle\text{-}edges\ (enum\text{-}cycle\ j))$
proof
assume $*$: $(source\ \Delta,\ y) \in set\ (cycle\text{-}edges\ (enum\text{-}cycle\ j))$
have *cycle*: $cycle\ \Delta\ (enum\text{-}cycle\ j)$ **using** *assms* **by**(*rule cycle-enum-cycle*)
from *cycle-leave-ex-enter*[*OF this* $*$]
obtain *z* **where** $(z,\ source\ \Delta) \in set\ (cycle\text{-}edges\ (enum\text{-}cycle\ j))$ **..**
with *cycle-edges-edges*[*OF cycle*] **have** $(z,\ source\ \Delta) \in \mathbf{E}$ **..**
thus *False* **using** *source-in*[*of z*] **by** *simp*
qed
then show *?case* **using** *Suc.IH* **by** *simp*
qed

lemma *h-minus-aux-cycle*:
fixes *j* **defines** $C \equiv enum\text{-}cycle\ j$
assumes $cycles\ \Delta \neq \{\}$
shows $\exists\, e{\in}set\ (cycle\text{-}edges\ C).\ h\text{-}minus\text{-}aux\ (Suc\ j)\ e = h'\ e$
proof −
let $?A = \{h'\ e' - h\text{-}minus\text{-}aux\ j\ e' | e'.\ e' \in set\ (cycle\text{-}edges\ C)\}$
from *assms* **have** $cycle\ \Delta\ C$ **by** *auto*
from *cycle-edges-not-Nil*[*OF this*] **have** $Min\ ?A \in ?A$ **using** [[*simproc add*: *finite-Collect*]]
by(*intro Min-in*)(*fastforce simp add*: *neq-Nil-conv*)+
then obtain e' **where** *e*: $e' \in set\ (cycle\text{-}edges\ C)$

and *Min ?A* = *h′ e′* − *h-minus-aux j e′* **by** *auto*
hence *h-minus-aux* (*Suc j*) *e′* = *h′ e′*
by(*simp add*: *C-def h-minus-aux-le-h′*)
with *e* **show** *?thesis* **by** *blast*
qed

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* (*λe′*. *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(*induction i e rule*: *h-minus.induct*)
case *0*: (*1 e*) **show** *?case* **by** *simp*
next
case *Suc*: (*2 i e*)
note *IH* = *Suc.IH*(*2*)[*OF UNIV-I*]
let *?h′* = *λe′*. *h-plus* (*Suc i*) *e′* − *h-minus i e′*
have *h′*: *?h′ e′* ≠ *top* **for** *e′* **using** *IH*(*1*)[*of e′*] **by** *simp*

have (⨆*j*. *h-minus-aux ?h′ j e*) ≤ *?h′ e* **by**(*rule SUP-least*)(*rule h-minus-aux-le-h′*[*OF h′*])
hence *h-minus* (*Suc i*) *e* ≤ *h-minus i e* + ... **by**(*simp add*: *add-mono*)
also have ... = *h-plus* (*Suc i*) *e* **using** *IH*[*of e*] *h-plus-mono*[*of i e*]
by *auto*
finally show *?case* **.**
qed

lemma *finite-h′*: *h-plus* (*Suc i*) *e* − *h-minus i e* ≠ *top*
by *simp*

lemma *h-minus-mono*: *h-minus i e* ≤ *h-minus* (*Suc i*) *e*
proof −
have *h-minus i e* + *0* ≤ *h-minus* (*Suc i*) *e* **unfolding** *h-minus.simps*
by(*rule add-mono*; *simp add*: *SUP-upper2*)
thus *?thesis* **by** *simp*
qed

lemma *h-minus-finite* [*simp*]: *h-minus i e* ≠ ⊤
proof −
have *h-minus i e* ≤ *h-plus i e* **by**(*rule h-minus-le-h-plus*)
also have ... < ⊤ **by** (*simp add*: *less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

lemma *d-OUT-h-minus*:

assumes *cycles*: *cycles* Δ ≠ {}
shows *d-OUT* (*h-minus i*) *x* = *d-IN* (*h-minus i*) *x*
proof(*induction i*)
case (*Suc i*)
let *?h′* = *λe. h-plus* (*Suc i*) *e* − *h-minus i e*
have *d-OUT* (*λe. h-minus* (*Suc i*) *e*) *x* = *d-OUT* (*h-minus i*) *x* + *d-OUT* (*λe. SUP j. h-minus-aux ?h′ j e*) *x*
by(*simp add*: *d-OUT-add SUP-upper2*)
also have *d-OUT* (*λe. SUP j. h-minus-aux ?h′ j e*) *x* = (*SUP j. d-OUT* (*h-minus-aux ?h′ j*) *x*)
by(*rule d-OUT-monotone-convergence-SUP incseq-SucI le-funI h-minus-aux-mono finite-h′*)+
also have . . . = (*SUP j. d-IN* (*h-minus-aux ?h′ j*) *x*)
by(*rule SUP-cong*[*OF refl*])(*rule d-OUT-h-minus-aux*[*OF finite-h′ cycles*])
also have . . . = *d-IN* (*λe. SUP j. h-minus-aux ?h′ j e*) *x*
by(*rule d-IN-monotone-convergence-SUP*[*symmetric*] *incseq-SucI le-funI h-minus-aux-mono finite-h′*)+
also have *d-OUT* (*h-minus i*) *x* + . . . = *d-IN* (*λe. h-minus* (*Suc i*) *e*) *x* **using** *Suc.IH*
by(*simp add*: *d-IN-add SUP-upper2*)
finally show *?case* **.**
qed *simp*

lemma *h-minus-source*:
assumes *cycles* Δ ≠ {}
shows *h-minus n* (*source* Δ, *y*) = *0*
by(*induction n*)(*simp-all add*: *h-minus-aux-source*[*OF finite-h′ assms*])

lemma *h-minus-source-in* [*simp*]: *h-minus i* (*x*, *source* Δ) = *0*
using *h-minus-le-h-plus*[*of i* (*x*, *source* Δ)] **by** *simp*

lemma *h-minus-OUT-finite* [*simp*]: *d-OUT* (*h-minus i*) *x* ≠ *top*
proof −
have *d-OUT* (*h-minus i*) *x* ≤ *d-OUT* (*h-plus i*) *x* **by**(*rule d-OUT-mono*)(*rule h-minus-le-h-plus*)
also have . . . < ⊤ **by** (*simp add*: *less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

lemma *h-minus-cycle*:
assumes *cycle* Δ *C*
shows ∃ *e*∈*set* (*cycle-edges C*). *h-minus i e* = *h-plus i e*
proof(*cases i*)
case (*Suc i*)
let *?h′* = *λe. h-plus* (*Suc i*) *e* − *h-minus i e*
from *assms* **have** *cycles*: *cycles* Δ ≠ {} **by** *auto*
with *assms from-nat-into-surj*[*of cycles* Δ *C*] **obtain** *j* **where** *j*: *C* = *enum-cycle j*
by(*auto simp add*: *enum-cycle-def*)

from *h-minus-aux-cycle*[*of ?h′ j*, *OF finite-h′ cycles*] *j*
obtain *e* **where** *e*: $e \in$ *set* (*cycle-edges C*) **and** *h-minus-aux ?h′* (*Suc j*) *e* = *?h′ e* **by**(*auto*)
then have *h-plus* (*Suc i*) *e* = *h-minus i e* + *h-minus-aux ?h′* (*Suc j*) *e*
using *order-trans*[*OF h-minus-le-h-plus h-plus-mono*]
by (*subst eq-commute*) *simp*
also have ... $\leq$ *h-minus* (*Suc i*) *e* **unfolding** *h-minus.simps*
by(*intro add-mono SUP-upper*; *simp*)
finally show *?thesis* **using** *e h-minus-le-h-plus*[*of Suc i e*] *Suc* **by** *auto*
next
case *0*
from *cycle-edges-not-Nil*[*OF assms*] **obtain** *x y* **where** *e*: $(x, y) \in$ *set* (*cycle-edges C*)
by(*fastforce simp add*: *neq-Nil-conv*)
then have $x \neq$ *source* Δ **using** *assms* **by**(*auto dest*: *source-out-not-cycle*)
hence *h-plus 0* (*x*, *y*) = *0* **by** *simp*
with *e 0* **show** *?thesis* **by**(*auto simp del*: *h-plus.simps*)
qed

abbreviation *lim-h-plus* :: *′v edge* $\Rightarrow$ *ennreal*
where *lim-h-plus e* $\equiv$ *SUP n. h-plus n e*

abbreviation *lim-h-minus* :: *′v edge* $\Rightarrow$ *ennreal*
where *lim-h-minus e* $\equiv$ *SUP n. h-minus n e*

lemma *lim-h-plus-le-g*: *lim-h-plus e* $\leq$ *g e*
by(*rule SUP-least*)(*rule h-plus-le-g*)

lemma *lim-h-plus-finite* [*simp*]: *lim-h-plus e* $\neq$ *top*
proof −
have *lim-h-plus e* $\leq$ *g e* **by**(*rule lim-h-plus-le-g*)
also have ... < *top* **by** (*simp add*: *less-top*[*symmetric*])
finally show *?thesis* **unfolding** *less-top* **.**
qed

lemma *lim-h-minus-le-lim-h-plus*: *lim-h-minus e* $\leq$ *lim-h-plus e*
by(*rule SUP-mono*)(*blast intro*: *h-minus-le-h-plus*)

lemma *lim-h-minus-finite* [*simp*]: *lim-h-minus e* $\neq$ *top*
proof −
have *lim-h-minus e* $\leq$ *lim-h-plus e* **by**(*rule lim-h-minus-le-lim-h-plus*)
also have ... < *top* **unfolding** *less-top*[*symmetric*] **by** (*rule lim-h-plus-finite*)
finally show *?thesis* **unfolding** *less-top*[*symmetric*] **by** *simp*
qed

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

```
  have d-IN lim-h-minus x ≤ d-IN lim-h-plus x
    by(intro d-IN-mono le-funI lim-h-minus-le-lim-h-plus)
  also have ... ≤ d-IN g x by(intro d-IN-mono le-funI lim-h-plus-le-g)
  also have ... < ⊤ using assms by(simp add: finite-IN-g less-top[symmetric])
  finally show ?thesis by simp
qed

lemma lim-h-plus-OUT-IN:
  assumes x ≠ source Δ x ≠ sink Δ
  shows d-OUT lim-h-plus x = d-IN lim-h-plus x
proof(cases x ∈ V)
  case True
  have d-OUT lim-h-plus x = (SUP n. d-OUT (h-plus n) x)
    by(rule d-OUT-monotone-convergence-SUP incseq-SucI le-funI h-plus-mono)+
  also have ... = (SUP n. d-IN (h-plus n) x) (is ?lhs = ?rhs)
  proof(rule antisym)
   show ?lhs ≤ ?rhs by(rule SUP-mono)(auto intro: h-plus-OUT-le-IN[OF assms(1)])
    show ?rhs ≤ ?lhs
    proof(rule SUP-mono)
      fix i
      from enum-v-repeat[OF True assms, of i]
      obtain i' where i' > i enum-v i' = x by auto
      moreover then obtain i'' where i': i' = Suc i'' by(cases i') auto
       ultimately have d-OUT (h-plus i') x = d-IN (h-plus i'') x using ‹x ≠
source Δ›
        by(simp add: h-plus-OUT-eq-IN)
      moreover have i ≤ i'' using ‹i < i'› i' by simp
        then have d-IN (h-plus i) x ≤ d-IN (h-plus i'') x by(intro d-IN-mono
h-plus-mono')
      ultimately have d-IN (h-plus i) x ≤ d-OUT (h-plus i') x by simp
      thus ∃i'∈UNIV. d-IN (h-plus i) x ≤ d-OUT (h-plus i') x by blast
    qed
  qed
  also have ... = d-IN lim-h-plus x
   by(rule d-IN-monotone-convergence-SUP[symmetric] incseq-SucI le-funI h-plus-mono)+
  finally show ?thesis .
next
  case False
  have (x, y) ∉ support-flow lim-h-plus for y using False h-plus-outside[of (x, y)]
    by(fastforce elim!: support-flow.cases simp add: less-SUP-iff vertex-def)
 moreover have (y, x) ∉ support-flow lim-h-plus for y using False h-plus-outside[of
(y, x)]
    by(fastforce elim!: support-flow.cases simp add: less-SUP-iff vertex-def)
  ultimately show ?thesis
   by(auto simp add: d-OUT-alt-def2 d-IN-alt-def2 AE-count-space intro!: nn-integral-cong-AE)
qed

lemma lim-h-minus-OUT-IN:
  assumes cycles: cycles Δ ≠ {}
```

shows *d-OUT lim-h-minus x* = *d-IN lim-h-minus x*
proof −
have *d-OUT lim-h-minus x* = (*SUP n. d-OUT* (*h-minus n*) *x*)
by(*rule d-OUT-monotone-convergence-SUP incseq-SucI le-funI h-minus-mono*)+
also have ... = (*SUP n. d-IN* (*h-minus n*) *x*) **using** *cycles* **by**(*simp add*: *d-OUT-h-minus*)
also have ... = *d-IN lim-h-minus x*
by(*rule d-IN-monotone-convergence-SUP*[*symmetric*] *incseq-SucI le-funI h-minus-mono*)+
finally show *?thesis* **.**
qed

definition *h* :: *'v edge* ⇒ *ennreal*
where *h e* = *lim-h-plus e* − (*if cycles* Δ ≠ {} *then lim-h-minus e else 0*)

lemma *h-le-lim-h-plus*: *h e* ≤ *lim-h-plus e*
by (*simp add*: *h-def*)

lemma *h-le-g*: *h e* ≤ *g e*
using *h-le-lim-h-plus*[*of e*] *lim-h-plus-le-g*[*of e*] **by** *simp*

lemma *flow-h*: *flow* Δ *h*
proof
fix *e*
have *h e* ≤ *lim-h-plus e* **by**(*rule h-le-lim-h-plus*)
also have ... ≤ *g e* **by**(*rule lim-h-plus-le-g*)
also have ... ≤ *capacity* Δ *e* **using** *g* **by**(*rule flowD-capacity*)
finally show *h e* ≤ ... **.**
next
fix *x*
assume *x* ≠ *source* Δ *x* ≠ *sink* Δ
then show *KIR h x*
by (*cases cycles* Δ = {})
(*auto simp add*: *h-def*[*abs-def*] *lim-h-plus-OUT-IN d-OUT-diff d-IN-diff lim-h-minus-le-lim-h-plus lim-h-minus-OUT-IN*)
qed

lemma *value-h-plus*: *value-flow* Δ (*h-plus i*) = *value-flow* Δ *g* (**is** *?lhs* = *?rhs*)
proof(*rule antisym*)
show *?lhs* ≤ *?rhs* **by**(*rule d-OUT-mono*)(*rule h-plus-le-g*)

have *?rhs* ≤ *value-flow* Δ (*h-plus 0*)
by(*auto simp add*: *d-OUT-def cong*: *if-cong intro*!: *nn-integral-mono*)
also have ... ≤ *value-flow* Δ (*h-plus i*)
by(*rule d-OUT-mono*)(*rule h-plus-mono'*; *simp*)
finally show *?rhs* ≤ *?lhs* **.**
qed

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

have *?lhs* $\leq$ *value-flow* Δ *lim-h-plus* **using** *ennreal-minus-mono*
by(*fastforce simp add*: *h-def intro*!: *d-OUT-mono*)
also have ... $\leq$ *?rhs* **by**(*rule d-OUT-mono*)(*rule lim-h-plus-le-g*)
finally show *?lhs* $\leq$ *?rhs* **.**

show *?rhs* $\leq$ *?lhs*
by(*auto simp add*: *d-OUT-def h-def h-minus-source cong*: *if-cong intro*!: *nn-integral-mono SUP-upper2*[**where** *i=0*])
qed

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*
by(*simp add*: *d-IN-def h-diff-def cong del*: *if-weak-cong*)

lemma *h-diff-le-h-plus*: *h-diff i e* $\leq$ *h-plus i e*
by(*simp add*: *h-diff-def*)

lemma *h-diff-le-g*: *h-diff i e* $\leq$ *g e*
using *h-diff-le-h-plus*[*of i e*] *h-plus-le-g*[*of i e*] **by** *simp*

lemma *h-diff-loop* [*simp*]: *h-diff i* (*x*, *x*) = *0*
using *h-diff-le-g*[*of i* (*x*, *x*)] **by** *simp*

lemma *supp-h-diff-edges*: *support-flow* (*h-diff i*) $\subseteq$ **E**
proof
fix *e*
assume *e* $\in$ *support-flow* (*h-diff i*)
then have *0* < *h-diff i e* **by**(*auto elim*: *support-flow.cases*)
also have *h-diff i e* $\leq$ *h-plus i e* **by**(*rule h-diff-le-h-plus*)
finally show *e* $\in$ **E** **using** *h-plus-outside*[*of e i*] **by**(*cases e* $\in$ **E**) *auto*
qed

lemma *h-diff-OUT-le-IN*:
assumes *x* $\neq$ *source* Δ
shows *d-OUT* (*h-diff i*) *x* $\leq$ *d-IN* (*h-diff i*) *x*
proof(*cases cycles* $\Delta \neq \{\}$)
case *False*
thus *?thesis* **using** *assms* **by**(*simp add*: *h-diff-def*[*abs-def*] *h-plus-OUT-le-IN*)
next
case *cycles*: *True*
then have *d-OUT* (*h-diff i*) *x* = *d-OUT* (*h-plus i*) *x* − *d-OUT* (*h-minus i*) *x*
unfolding *h-diff-def*[*abs-def*] **using** *assms*
by (*simp add*: *h-minus-le-h-plus d-OUT-diff*)
also have ... $\leq$ *d-IN* (*h-plus i*) *x* − *d-IN* (*h-minus i*) *x* **using** *cycles assms*
by(*intro ennreal-minus-mono h-plus-OUT-le-IN*)(*simp-all add*: *d-OUT-h-minus*)
also have ... = *d-IN* (*h-diff i*) *x* **using** *cycles*

unfolding *h-diff-def*[*abs-def*] **by**(*subst d-IN-diff*)(*simp-all add*: *h-minus-le-h-plus d-OUT-h-minus*[*symmetric*])
finally show *?thesis* **.**
qed

lemma *h-diff-cycle*:
assumes *cycle* Δ *p*
shows $\exists$ *e*$\in$*set* (*cycle-edges p*). *h-diff i e* = *0*
proof −
from *h-minus-cycle*[*OF assms*, *of i*] **obtain** *e*
where *e*: *e* $\in$ *set* (*cycle-edges p*) **and** *h-minus i e* = *h-plus i e* **by** *auto*
hence *h-diff i e* = *0* **using** *assms* **by**(*auto simp add*: *h-diff-def*)
with *e* **show** *?thesis* **by** *blast*
qed

lemma *d-IN-h-le-value'*: *d-IN* (*h-diff i*) *x* $\leq$ *value-flow* Δ (*h-plus i*)
proof −
let *?supp* = *support-flow* (*h-diff i*)
define *X* **where** *X* = {*y*. (*y*, *x*) $\in$ *?supp*^*} − {*x*}

{ **fix** *x y*
assume *x*: *x* $\notin$ *X* **and** *y*: *y* $\in$ *X*
{ **assume** *yx*: (*y*, *x*) $\in$ *?supp** **and** *neq*: *y* $\neq$ *x* **and** *xy*: (*x*, *y*) $\in$ *?supp*
from *yx* **obtain** *p'* **where** *rtrancl-path* (λ*x y*. (*x*, *y*) $\in$ *?supp*) *y p' x*
unfolding *rtrancl-def rtranclp-eq-rtrancl-path* **by** *auto*
then obtain *p* **where** *p*: *rtrancl-path* (λ*x y*. (*x*, *y*) $\in$ *?supp*) *y p x*
and *distinct*: *distinct* (*y* # *p*) **by**(*rule rtrancl-path-distinct*)
with *neq* **have** *p* $\neq$ [] **by**(*auto elim*: *rtrancl-path.cases*)

from *xy* **have** (*x*, *y*) $\in$ **E** **using** *supp-h-diff-edges*[*of i*] **by**(*auto*)
moreover from *p* **have** *path* Δ *y p x*
by(*rule rtrancl-path-mono*)(*auto dest*: *supp-h-diff-edges*[*THEN subsetD*])
ultimately have *path* Δ *x* (*y* # *p*) *x* **by**(*auto intro*: *rtrancl-path.intros*)
hence *cycle*: *cycle* Δ (*y* # *p*) **using** - *distinct* **by**(*rule cycle*) *simp*
from *h-diff-cycle*[*OF this*, *of i*] **obtain** *e*
where *e*: *e* $\in$ *set* (*cycle-edges* (*y* # *p*)) **and** *0*: *h-diff i e* = *0* **by** *blast*
from *e* **obtain** *n* **where** *e'*: *e* = ((*y* # *p*) ! *n*, (*p* @ [*y*]) ! *n*) **and** *n*: *n* < *Suc* (*length p*)
by(*auto simp add*: *cycle-edges-def set-zip*)
have *e* $\in$ *?supp*
proof(*cases n* = *length p*)
case *True*
with *rtrancl-path-last*[*OF p*] ‹*p* $\neq$ []› **have** (*y* # *p*) ! *n* = *x*
by(*cases p*)(*simp-all add*: *last-conv-nth del*: *last.simps*)
with *e' True* **have** *e* = (*x*, *y*) **by** *simp*
with *xy* **show** *?thesis* **by** *simp*
next
case *False*
with *n* **have** *n* < *length p* **by** *simp*

with *rtrancl-path-nth*[*OF p this*] *e′* **show** *?thesis* **by**(*simp add*: *nth-append*)
qed
with *0* **have** *False* **by**(*simp add*: *support-flow.simps*) **}**
hence (*x*, *y*) ∉ *?supp* **using** *x y*
by(*auto simp add*: *X-def intro*: *converse-rtrancl-into-rtrancl*)
then have *h-diff i* (*x*, *y*) = *0*
by(*simp add*: *support-flow.simps*) **}**
note *acyclic* = *this*

{ fix *y*
assume *y* ∉ *X*
hence (*y*, *x*) ∉ *?supp* **by**(*auto simp add*: *X-def support-flow.simps intro*: *not-in-support-flowD*)
hence *h-diff i* (*y*, *x*) = *0* **by**(*simp add*: *support-flow.simps*) **}**
note *in-X* = *this*

let *?diff* = λ*x*. (∑⁺ *y*. *h-diff i* (*x*, *y*) ∗ *indicator X x* ∗ *indicator X y*)
have *finite2*: (∑⁺ *x*. *?diff x*) ≠ *top* (**is** *?lhs* ≠ -)
proof −
have *?lhs* ≤ (∑⁺ *x*∈*UNIV*. ∑⁺ *y*. *h-plus i* (*x*, *y*))
by(*intro nn-integral-mono*)(*auto simp add*: *h-diff-def split*: *split-indicator*)
also have … = (∑⁺ *e*. *h-plus i e*) **by**(*rule nn-integral-fst-count-space*)
also have … < ⊤ **by**(*simp add*: *h-plus-sum-finite less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
qed
have *finite1*: *?diff x* ≠ *top* **for** *x*
using *finite2* **by**(*rule neq-top-trans*)(*rule nn-integral-ge-point*, *simp*)
have *finite3*: (∑⁺ *x*. *d-OUT* (*h-diff i*) *x* ∗ *indicator* (*X* − {*source* Δ}) *x*) ≠ ⊤ (**is** *?lhs* ≠ -)
proof −
have *?lhs* ≤ (∑⁺ *x*∈*UNIV*. ∑⁺ *y*. *h-plus i* (*x*, *y*)) **unfolding** *d-OUT-def*
apply(*simp add*: *nn-integral-multc*[*symmetric*])
apply(*intro nn-integral-mono*)
apply(*auto simp add*: *h-diff-def split*: *split-indicator*)
done
also have … = (∑⁺ *e*. *h-plus i e*) **by**(*rule nn-integral-fst-count-space*)
also have … < ⊤ **by**(*simp add*: *h-plus-sum-finite less-top*[*symmetric*])
finally show *?thesis* **by** *simp*
qed

have *d-IN* (*h-diff i*) *x* = (∑⁺ *y*. *h-diff i* (*y*, *x*) ∗ *indicator X y*) **unfolding** *d-IN-def*
by(*rule nn-integral-cong*)(*simp add*: *in-X split*: *split-indicator*)
also have … ≤ (∑⁺ *x*∈− *X*. ∑⁺ *y*. *h-diff i* (*y*, *x*) ∗ *indicator X y*)
by(*rule nn-integral-ge-point*)(*simp add*: *X-def*)
also have … = (∑⁺ *x*∈*UNIV*. ∑⁺ *y*. *h-diff i* (*y*, *x*) ∗ *indicator X y* ∗ *indicator* (− *X*) *x*)
by(*simp add*: *nn-integral-multc nn-integral-count-space-indicator*)
also have … = (∑⁺ *x*∈*UNIV*. ∑⁺ *y*. *h-diff i* (*x*, *y*) ∗ *indicator X x* ∗ *indicator*

$(- X)$ y)
by(*subst nn-integral-snd-count-space*[**where** *f*=*case-prod* -, *simplified*])(*simp add*: *nn-integral-fst-count-space*[**where** *f*=*case-prod* -, *simplified*])
also have ... = ($\sum^+$ *x*∈*UNIV*. ($\sum^+$ *y*. *h-diff i* (*x*, *y*) ∗ *indicator X x* ∗ *indicator* (− *X*) *y*) + (*?diff x* − *?diff x*))
by(*simp add*: *finite1*)
also have ... = ($\sum^+$ *x*∈*UNIV*. ($\sum^+$ *y*. *h-diff i* (*x*, *y*) ∗ *indicator X x* ∗ *indicator* (− *X*) *y* + *h-diff i* (*x*, *y*) ∗ *indicator X x* ∗ *indicator X y*) − *?diff x*)
apply (*subst add-diff-eq-ennreal*)
apply *simp*
by(*subst nn-integral-add*[*symmetric*])(*simp-all add*:)
also have ... = ($\sum^+$ *x*∈*UNIV*. ($\sum^+$ *y*. *h-diff i* (*x*, *y*) ∗ *indicator X x*) − *?diff x*)
by(*auto intro*!: *nn-integral-cong arg-cong2*[**where** *f*=(−)] *split*: *split-indicator*)
also have ... = ($\sum^+$ *x*∈*UNIV*. $\sum^+$ *y*∈*UNIV*. *h-diff i* (*x*, *y*) ∗ *indicator X x*) − ($\sum^+$ *x*. *?diff x*)
by(*subst nn-integral-diff*)(*auto simp add*: *AE-count-space finite2 intro*!: *nn-integral-mono split*: *split-indicator*)
also have ($\sum^+$ *x*∈*UNIV*. $\sum^+$ *y*∈*UNIV*. *h-diff i* (*x*, *y*) ∗ *indicator X x*) = ($\sum^+$ *x*. *d-OUT* (*h-diff i*) *x* ∗ *indicator X x*)
unfolding *d-OUT-def* **by**(*simp add*: *nn-integral-multc*)
also have ... = ($\sum^+$ *x*. *d-OUT* (*h-diff i*) *x* ∗ *indicator* (*X* − {*source* Δ}) *x* + *value-flow* Δ (*h-diff i*) ∗ *indicator X* (*source* Δ) ∗ *indicator* {*source* Δ} *x*)
by(*rule nn-integral-cong*)(*simp split*: *split-indicator*)
also have ... = ($\sum^+$ *x*. *d-OUT* (*h-diff i*) *x* ∗ *indicator* (*X* − {*source* Δ}) *x*) + *value-flow* Δ (*h-diff i*) ∗ *indicator X* (*source* Δ)
(**is** - = *?out* **is** - = - + *?value*)
by(*subst nn-integral-add*) *simp-all*
also have ($\sum^+$ *x*∈*UNIV*. $\sum^+$ *y*. *h-diff i* (*x*, *y*) ∗ *indicator X x* ∗ *indicator X y*) =
($\sum^+$ *x*∈*UNIV*. $\sum^+$ *y*. *h-diff i* (*x*, *y*) ∗ *indicator X y*)
using *acyclic* **by**(*intro nn-integral-cong*)(*simp split*: *split-indicator*)
also have ... = ($\sum^+$ *y*∈*UNIV*. $\sum^+$ *x*. *h-diff i* (*x*, *y*) ∗ *indicator X y*)
by(*subst nn-integral-snd-count-space*[**where** *f*=*case-prod* -, *simplified*])(*simp add*: *nn-integral-fst-count-space*[**where** *f*=*case-prod* -, *simplified*])
also have ... = ($\sum^+$ *y*. *d-IN* (*h-diff i*) *y* ∗ *indicator X y*) **unfolding** *d-IN-def*
by(*simp add*: *nn-integral-multc*)
also have ... = ($\sum^+$ *y*. *d-IN* (*h-diff i*) *y* ∗ *indicator* (*X* − {*source* Δ}) *y*)
by(*rule nn-integral-cong*)(*simp split*: *split-indicator*)
also have *?out* − ... ≤ ($\sum^+$ *x*. *d-OUT* (*h-diff i*) *x* ∗ *indicator* (*X* − {*source* Δ}) *x*) − ... + *?value*
by (*auto simp add*: *add-ac intro*!: *add-diff-le-ennreal*)
also have ... ≤ *0* + *?value* **using** *h-diff-OUT-le-IN finite3*
by(*intro nn-integral-mono add-right-mono*)(*auto split*: *split-indicator intro*!: *diff-eq-0-ennreal nn-integral-mono simp add*: *less-top*)
also have ... ≤ *value-flow* Δ (*h-diff i*) **by**(*simp split*: *split-indicator*)
also have ... ≤ *value-flow* Δ (*h-plus i*) **by**(*rule d-OUT-mono le-funI h-diff-le-h-plus*)+
finally show *?thesis* **.**
qed

lemma *d-IN-h-le-value*: *d-IN h x* $\leq$ *value-flow* Δ *h* (**is** *?lhs* $\leq$ *?rhs*)
proof $-$
  have [*tendsto-intros*]: (λ*i. h-plus i e*) $\longrightarrow$ *lim-h-plus e* **for** *e*
    by(*rule LIMSEQ-SUP incseq-SucI h-plus-mono*)+
  have [*tendsto-intros*]: (λ*i. h-minus i e*) $\longrightarrow$ *lim-h-minus e* **for** *e*
    by(*rule LIMSEQ-SUP incseq-SucI h-minus-mono*)+
  have (λ*i. h-diff i e*) $\longrightarrow$ *lim-h-plus e* $-$ (*if cycles* $\Delta \neq \{\}$ *then lim-h-minus e else 0*) **for** *e*
    by(*auto intro*!: *tendsto-intros tendsto-diff-ennreal simp add*: *h-diff-def simp del*: *Sup-eq-top-iff SUP-eq-top-iff*)
  then have *d-IN h x* = ($\sum^{+}$ *y. liminf* (λ*i. h-diff i* (*y*, *x*)))
    by(*simp add*: *d-IN-def h-def tendsto-iff-Liminf-eq-Limsup*)
  also have ... $\leq$ *liminf* (λ*i. d-IN* (*h-diff i*) *x*) **unfolding** *d-IN-def*
    by(*rule nn-integral-liminf*) *simp-all*
  also have ... $\leq$ *liminf* (λ*i. value-flow* Δ *h*) **using** *d-IN-h-le-value'*[*of - x*]
    by(*intro Liminf-mono eventually-sequentiallyI*)(*auto simp add*: *value-h-plus value-h*)
  also have ... = *value-flow* Δ *h* **by**(*simp add*: *Liminf-const*)
  finally show *?thesis* **.**
qed

lemma *flow-cleanup*: — Lemma 5.4
  $\exists$ *h* $\leq$ *g. flow* Δ *h* $\wedge$ *value-flow* Δ *h* = *value-flow* Δ *g* $\wedge$ ($\forall$ *x. d-IN h x* $\leq$ *value-flow* Δ *h*)
by(*intro exI*[**where** *x*=*h*] *conjI strip le-funI d-IN-h-le-value flow-h value-h h-le-g*)

end

9.2 Residual network

context *countable-network* **begin**

definition *residual-network* :: *'v flow* $\Rightarrow$ (*'v*, *'more*) *network-scheme*
where *residual-network f* =
  ⦇*edge* = λ*x y. edge* Δ *x y* $\vee$ *edge* Δ *y x* $\wedge$ *y* $\neq$ *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
by(*simp-all add*: *residual-network-def*)

lemma **E**-*residual-network*: $\mathbf{E}_{\mathit{residual\text{-}network}\ f} = \mathbf{E} \cup \{(x, y).\ (y, x) \in \mathbf{E} \wedge y \neq$ *source* $\Delta\}$
by *auto*

lemma *vertices-residual-network* [*simp*]: *vertex* (*residual-network f*) = *vertex* Δ
by(*auto simp add*: *vertex-def fun-eq-iff*)

inductive *wf-residual-network* :: *bool*
where $[\![\ \bigwedge x\ y.\ (x, y) \in \mathbf{E} \Longrightarrow (y, x) \notin \mathbf{E};\ (\mathit{source}\ \Delta, \mathit{sink}\ \Delta) \notin \mathbf{E}\ ]\!] \Longrightarrow$ *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*
by(*auto simp add*: *wf-residual-network.simps*)

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
  have *countable* (*converse* **E**) **by** *simp*
  then have *countable* $\{(x, y).\ (y, x) \in \mathbf{E} \wedge y \neq \mathit{source}\ \Delta\}$
    by(*rule countable-subset*[*rotated*]) *auto*
  then show *countable* $\mathbf{E}_{?\Delta}$ **unfolding** **E**-*residual-network* **by** *simp*

  show *source ?*Δ ≠ *sink ?*Δ **by** *simp*
  show *capacity ?*Δ *e* = *0* **if** $e \notin \mathbf{E}_{?\Delta}$ **for** *e* **using** *that* **by**(*cases e*)(*auto intro*: *flowD-outside*[*OF f*])
  show *capacity ?*Δ *e* ≠ *top* **for** *e*
    using *flowD-finite*[*OF f*] **by**(*cases e*) *auto*
qed

end

context *antiparallel-edges* **begin**

interpretation Δ′′: *countable-network* Δ′′ **by**(*rule* Δ′′-*countable-network*)

lemma Δ′′-*flow-attainability*:
  assumes *flow-attainability-axioms* Δ
  shows *flow-attainability* Δ′′
proof −
  interpret *flow-attainability* Δ **using** - *assms* **by**(*rule flow-attainability.intro*) *unfold-locales*
  show *?thesis*
  proof
    show *d-IN* (*capacity* Δ′′) *v* ≠ ⊤ ∨ *d-OUT* (*capacity* Δ′′) *v* ≠ ⊤ **if** *v* ≠ *sink*

Δ'' **for** v
using *that finite-capacity* **by**(*cases v*)(*simp-all add*: *max-def*)
show $\neg$ *edge* Δ'' *v v* **for** *v* **by**(*auto elim*: *edg.cases*)
show $\neg$ *edge* Δ'' *v* (*source* Δ'') **for** *v* **by**(*simp add*: *source-in*)
qed
qed

lemma Δ''-*wf-residual-network*:
assumes *no-loop*: $\bigwedge x.$ $\neg$ *edge* Δ *x x*
shows Δ''.*wf-residual-network*
by(*auto simp add*: Δ''.*wf-residual-network.simps assms elim*!: *edg.cases*)

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)
proof −
interpret *res*: *countable-network residual-network f* **by**(*rule residual-countable-network*[*OF assms*])
show *?thesis*
proof
fix *x*
assume *sink*: $x \neq$ *sink ?*Δ
then consider (*source*) $x =$ *source* Δ | (*IN*) *d-IN* (*capacity* Δ) $x \neq \top$ | (*OUT*) $x \neq$ *source* Δ *d-OUT* (*capacity* Δ) $x \neq \top$
using *finite-capacity*[*of x*] **by** *auto*
then show *d-IN* (*capacity ?*Δ) $x \neq \top \vee$ *d-OUT* (*capacity ?*Δ) $x \neq \top$
proof(*cases*)
case *source*
hence *d-IN* (*capacity ?*Δ) $x = 0$ **by**(*simp add*: *d-IN-def source-in*)
thus *?thesis* **by** *simp*
next
case *IN*
have *d-IN* (*capacity ?*Δ) $x =$
($\sum^+$ *y*. (*capacity* Δ (*y*, *x*) − *f* (*y*, *x*)) ∗ *indicator* **E** (*y*, *x*) +
(*if* $x =$ *source* Δ *then 0 else f* (*x*, *y*) ∗ *indicator* **E** (*x*, *y*)))
using *flowD-outside*[*OF f*] **unfolding** *d-IN-def*
by(*auto intro*!: *nn-integral-cong split*: *split-indicator dest*: *wf-residual-networkD*[*OF wf*])
also have ... = ($\sum^+$ *y*. (*capacity* Δ (*y*, *x*) − *f* (*y*, *x*)) ∗ *indicator* **E** (*y*, *x*)) +
($\sum^+$ *y*. (*if* $x =$ *source* Δ *then 0 else f* (*x*, *y*) ∗ *indicator* **E** (*x*, *y*)))
(**is** - = *?in* + *?out*)

by(*subst nn-integral-add*)(*auto simp add*: *AE-count-space split*: *split-indicator intro*!: *flowD-capacity*[*OF f*])
also have ... $\leq$ *d-IN* (*capacity* Δ) *x* + (*if x* = *source* Δ *then 0 else d-OUT f x*) (**is** - $\leq$ *?in* + *?rest*)
unfolding *d-IN-def d-OUT-def*
by(*rule add-mono*)(*auto intro*!: *nn-integral-mono split*: *split-indicator simp add*: *nn-integral-0-iff-AE AE-count-space intro*!: *diff-le-self-ennreal*)
also consider (*source*) *x* = *source* Δ | (*inner*) *x* $\neq$ *source* Δ *x* $\neq$ *sink* Δ **using** *sink* **by** *auto*
then have *?rest* < $\top$
proof *cases*
case *inner*
show *?thesis* **using** *inner flowD-finite-OUT*[*OF f inner*] **by** (*simp add*: *less-top*)
qed *simp*
ultimately show *?thesis* **using** *IN sink* **by** (*auto simp*: *less-top*[*symmetric*] *top-unique*)
next
case *OUT*
have *d-OUT* (*capacity ?*Δ) *x* =
($\sum^+$ *y*. (*capacity* Δ (*x*, *y*) $-$ *f* (*x*, *y*)) $*$ *indicator* **E** (*x*, *y*) +
(*if y* = *source* Δ *then 0 else f* (*y*, *x*) $*$ *indicator* **E** (*y*, *x*)))
using *flowD-outside*[*OF f*] **unfolding** *d-OUT-def*
by(*auto intro*!: *nn-integral-cong split*: *split-indicator dest*: *wf-residual-networkD*[*OF wf*] *simp add*: *source-in*)
also have ... = ($\sum^+$ *y*. (*capacity* Δ (*x*, *y*) $-$ *f* (*x*, *y*)) $*$ *indicator* **E** (*x*, *y*)) +
($\sum^+$ *y*. (*if y* = *source* Δ *then 0 else f* (*y*, *x*) $*$ *indicator* **E** (*y*, *x*)))
(**is** - = *?in* + *?out*)
by(*subst nn-integral-add*)(*auto simp add*: *AE-count-space split*: *split-indicator intro*!: *flowD-capacity*[*OF f*])
also have ... $\leq$ *d-OUT* (*capacity* Δ) *x* + *d-IN f x* (**is** - $\leq$ *?out* + *?rest*)
unfolding *d-IN-def d-OUT-def*
by(*rule add-mono*)(*auto intro*!: *nn-integral-mono split*: *split-indicator simp add*: *nn-integral-0-iff-AE AE-count-space intro*!: *diff-le-self-ennreal*)
also have *?rest* = *d-OUT f x* **using** *flowD-KIR*[*OF f OUT*(*1*)] *sink* **by** *simp*
also have *?out* + ... $\leq$ *?out* + *?out* **by**(*intro add-left-mono d-OUT-mono flowD-capacity*[*OF f*])
finally show *?thesis* **using** *OUT* **by** (*auto simp*: *top-unique*)
qed
next
show $\neg$ *edge ?*Δ *x x* **for** *x* **by**(*simp add*: *no-loop*)
show $\neg$ *edge ?*Δ *x* (*source ?*Δ) **for** *x* **by**(*simp add*: *source-in*)
qed
qed

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* = (λ(*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*)
by(*simp add*: *plus-flow-def*)

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

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* Δ *x y* $\Longrightarrow g\ (y, x) \leq f\ (x, y)$
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* **E** (*x*, *y*)) − ($\sum^+$ *y*. *g* (*y*, *x*) ∗ *indicator* **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* **E** (*y*, *x*)) − ($\sum^+$ *y*. *g* (*x*, *y*) ∗ *indicator* **E** (*y*, *x*))
(**is** - $\Longrightarrow$ *?IN* **is** - $\Longrightarrow$ - = - + *?g-in* − *?g-in'*)
proof −
assume *d-IN g x* $\neq$ *top*
then have *finite1*: ($\sum^+$ *y*. *g* (*y*, *x*) ∗ *indicator A* (*f y*)) $\neq$ *top* **for** *A f*
by(*rule neq-top-trans*)(*auto split*: *split-indicator simp add*: *d-IN-def intro*!: *nn-integral-mono*)

have *d-OUT* (*f* $\oplus$ *g*) *x* = ($\sum^+$ *y*. (*g* (*x*, *y*) + (*f* (*x*, *y*) − *g* (*y*, *x*))) ∗ *indicator* **E** (*x*, *y*))
unfolding *d-OUT-def* **by**(*rule nn-integral-cong*)(*simp split*: *split-indicator add*: *add-diff-eq-ennreal add.commute ennreal-diff-add-assoc g-le-f*)
also have ... = *?g-out* + ($\sum^+$ *y*. (*f* (*x*, *y*) − *g* (*y*, *x*)) ∗ *indicator* **E** (*x*, *y*))
(**is** - = - + *?rest*)
by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *AE-count-space g-le-f split*: *split-indicator intro*!: *nn-integral-cong*)
also have *?rest* = ($\sum^+$ *y*. *f* (*x*, *y*) ∗ *indicator* **E** (*x*, *y*)) − *?g-out'* (**is** - = *?f* − -)
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto intro*!: *nn-integral-cong split*: *split-indicator simp add*: *AE-count-space g-le-f finite1*)
done
also have *?f* = *d-OUT f x* **unfolding** *d-OUT-def* **using** *f-outside*
by(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ($\sum^+$ *y*. *g* (*x*, *y*) ∗ *indicator* **E** (*x*, *y*)) + (*d-OUT f x* − ($\sum^+$ *y*. *g* (*y*, *x*) ∗ *indicator* **E** (*x*, *y*))) =
d-OUT f x + *?g-out* − *?g-out'*
by (*subst ennreal-diff-add-assoc*[*symmetric*])
(*auto simp*: *ac-simps d-OUT-def intro*!: *nn-integral-mono g-le-f split*: *split-indicator*)

finally show *?OUT* **.**
next
assume *d-OUT g x* $\neq$ *top*
then have *finite2*: ($\sum^+$ *y. g* (*x*, *y*) $*$ *indicator A* (*f y*)) $\neq$ *top* **for** *A f*
by(*rule neq-top-trans*)(*auto split*: *split-indicator simp add*: *d-OUT-def intro*!: *nn-integral-mono*)

have *d-IN* (*f* $\oplus$ *g*) *x* = ($\sum^+$ *y.* (*g* (*y*, *x*) + (*f* (*y*, *x*) − *g* (*x*, *y*))) $*$ *indicator* **E** (*y*, *x*))
unfolding *d-IN-def* **by**(*rule nn-integral-cong*)(*simp split*: *split-indicator add*: *add-diff-eq-ennreal add.commute ennreal-diff-add-assoc g-le-f*)
also have ... = ($\sum^+$ *y*$\in$*UNIV. g* (*y*, *x*) $*$ *indicator* **E** (*y*, *x*)) + ($\sum^+$ *y.* (*f* (*y*, *x*) − *g* (*x*, *y*)) $*$ *indicator* **E** (*y*, *x*))
(**is** - = - + *?rest*)
by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *AE-count-space g-le-f split*: *split-indicator intro*!: *nn-integral-cong*)
also have *?rest* = ($\sum^+$ *y. f* (*y*, *x*) $*$ *indicator* **E** (*y*, *x*))− *?g-in′*
by(*subst nn-integral-diff*[*symmetric*])(*auto intro*!: *nn-integral-cong split*: *split-indicator simp add*: *add-ac add-diff-eq-ennreal AE-count-space g-le-f finite2*)
also have ($\sum^+$ *y. f* (*y*, *x*) $*$ *indicator* **E** (*y*, *x*)) = *d-IN f x*
unfolding *d-IN-def* **using** *f-outside* **by**(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ($\sum^+$ *y. g* (*y*, *x*) $*$ *indicator* **E** (*y*, *x*)) + (*d-IN f x* − ($\sum^+$ *y. g* (*x*, *y*) $*$ *indicator* **E** (*y*, *x*))) =
d-IN f x + *?g-in* − *?g-in′*
by (*subst ennreal-diff-add-assoc*[*symmetric*])
(*auto simp*: *ac-simps d-IN-def intro*!: *nn-integral-mono g-le-f split*: *split-indicator*)
finally show *?IN* **.**
qed

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* $\leq$ *d-IN f x* + *d-IN g x*
proof −
have *d-IN* (*f* $\oplus$ *g*) *x* $\leq$ ($\sum^+$ *y. f* (*y*, *x*) + *g* (*y*, *x*)) **unfolding** *d-IN-def*
by(*rule nn-integral-mono*)(*auto intro*: *diff-le-self-ennreal*)
also have ... = *d-IN f x* + *d-IN g x*
by(*subst nn-integral-add*)(*simp-all add*: *d-IN-def*)
finally show *?thesis* **.**
qed

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

proof(*intro flow.intros*)
  fix *e*
  from *c* **have** $c * f\ e \leq 1 * f\ e$ **by**(*rule mult-right-mono*) *simp*
  also have $\ldots \leq$ *capacity* Δ *e* **using** *flowD-capacity*[*OF f*, *of e*] **by** *simp*
  finally show $c * f\ e \leq \ldots$ **.**
next
  fix *x*
  assume *x*: $x \neq$ *source* Δ $x \neq$ *sink* Δ
  have *d-OUT* ($\lambda e.\ c * f\ e$) $x = c *$ *d-OUT f x* **by**(*simp add*: *d-OUT-cmult*)
  also have *d-OUT f x* = *d-IN f x* **using** *f x* **by**(*rule flowD-KIR*)
  also have $c * \ldots =$ *d-IN* ($\lambda e.\ c * f\ e$) *x* **by**(*simp add*: *d-IN-cmult*)
  finally show *KIR* ($\lambda e.\ c * f\ e$) *x* **.**
qed

lemma *value-scale-flow*:
  value-flow Δ ($\lambda e.\ c * f\ e$) $= c *$ *value-flow* Δ *f*
by(*rule d-OUT-cmult*)

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 −
  have *value-flow* Δ $f = (\sum^+ y \in$ **OUT** (*source* Δ). *f* (*source* Δ, *y*))
    by(*rule d-OUT-alt-def*)(*simp add*: *flowD-outside*[*OF f*])
  also have $\ldots = (\sum^+ y.\ f$ (*source* Δ, *y*) $*$ *indicator* $\{x\}$ *y*)
    by(*subst nn-integral-count-space-indicator*)(*auto intro*!: *nn-integral-cong split*: *split-indicator simp add*: *outgoing-def source-out*)
  also have $\ldots = f$ (*source* Δ, *x*) **by**(*simp add*: *one-ennreal-def*[*symmetric*] *max-def*)
  finally show *?thesis* **.**
qed

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 −
  interpret *RES*: *countable-network residual-network f* **using** *wf f* **by**(*rule residual-countable-network*)
  have *value-flow* Δ ($f \oplus g$) $= (\sum^+ y.\ f$ (*source* Δ, *y*) + *g* (*source* Δ, *y*))
    unfolding *d-OUT-def* **by**(*rule nn-integral-cong*)(*simp add*: *flowD-outside*[*OF f*] *RES.flowD-outside*[*OF g*] *source-in*)
  also have $\ldots =$ *value-flow* Δ *f* + *value-flow* Δ *g* **unfolding** *d-OUT-def*

by(*rule nn-integral-add*) *simp-all*
finally show *?thesis* **.**
qed

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
fix *e*
{ assume *e*: $e \in \mathbf{E}$
hence $(f \oplus g)\ e = f\ e + g\ e - g$ (*prod.swap e*) **by**(*cases e*) *simp*
also have $\ldots \leq f\ e + g\ e - 0$ **by**(*rule ennreal-minus-mono*) *simp-all*
also have $\ldots \leq f\ e +$ (*capacity* Δ $e - f\ e$)
using *e flowD-capacity*[*OF g*, *of e*] **by**(*simp split*: *prod.split-asm add*: *add-mono*)
also have $\ldots =$ *capacity* Δ *e* **using** *flowD-capacity*[*OF f*, *of e*]
by *simp*
also note *calculation* **}**
thus $(f \oplus g)\ e \leq$ *capacity* Δ *e* **by**(*cases e*) *auto*
next
fix *x*
assume *x*: $x \neq$ *source* Δ $x \neq$ *sink* Δ
have *g-le-f*: $g\ (y, x) \leq f\ (x, y)$ **if** *edge* Δ *x y* **for** *x y*
using *that flowD-capacity*[*OF g*, *of* (*y*, *x*)]
by(*auto split*: *if-split-asm dest*: *wf-residual-networkD*[*OF wf*] *elim*: *order-trans*)

interpret *RES*: *flow-attainability residual-network f* **using** *wf f* **by**(*rule residual-flow-attainability*)

have *finite1*: $(\sum^+ y.\ g\ (y, x) * \mathit{indicator}\ A\ (f\ y)) \neq \top$ **for** *A f*
using *RES.flowD-finite-IN*[*OF g*, *of x*]
by(*rule neq-top-trans*)(*auto simp add*: *x d-IN-def split*: *split-indicator intro*: *nn-integral-mono*)
have *finite2*: $(\sum^+ y.\ g\ (x, y) * \mathit{indicator}\ A\ (f\ y)) \neq \top$ **for** *A f*
using *RES.flowD-finite-OUT*[*OF g*, *of x*]
by(*rule neq-top-trans*)(*auto simp add*: *x d-OUT-def split*: *split-indicator intro*: *nn-integral-mono*)

have *d-OUT* $(f \oplus g)$ $x =$ *d-OUT f x* + $(\sum^+ y.\ g\ (x, y) * \mathit{indicator}\ \mathbf{E}\ (x, y)) - (\sum^+ y.\ g\ (y, x) * \mathit{indicator}\ \mathbf{E}\ (x, y))$
(**is** - = *?f* + *?g-out* − *?g-in*)
using *flowD-outside*[*OF f*] *g-le-f RES.flowD-finite-IN*[*OF g*, *of x*]
by(*rule OUT-plus-flow*)(*simp-all add*: *x*)
also have *?f* = *d-IN f x* **using** *f x* **by**(*auto dest*: *flowD-KIR*)
also have *?g-out* = $(\sum^+ y.\ g\ (x, y) * \mathit{indicator}\ (-\ \mathbf{E})\ (y, x))$
proof −
have $(\sum^+ y.\ g\ (x, y) * \mathit{indicator}\ (-\ \mathbf{E})\ (y, x)) =$
$(\sum^+ y.\ g\ (x, y) * \mathit{indicator}\ \mathbf{E}\ (x, y)) + (\sum^+ y.\ g\ (x, y) * \mathit{indicator}$ (−

E) $(x, y) *$ *indicator* $(-$ **E**$)$ $(y, x))$
by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *AE-count-space dest*: *wf-residual-networkD*[*OF wf*] *split*: *split-indicator intro*!: *nn-integral-cong*)
also have ($\sum^+$ *y*. *g* $(x, y) *$ *indicator* $(-$ **E**$)$ $(x, y) *$ *indicator* $(-$ **E**$)$ $(y, x))$ $=$ *0*
using *RES.flowD-outside*[*OF g*]
by(*auto simp add*: *nn-integral-0-iff-AE AE-count-space split*: *split-indicator*)
finally show *?thesis* **by** *simp*
qed
also have ... $=$ ($\sum^+$ *y*. *g* $(x, y) - g$ $(x, y) *$ *indicator* **E** $(y, x))$
by(*rule nn-integral-cong*)(*simp split*: *split-indicator add*: *RES.flowD-finite*[*OF g*])
also have ... $=$ *d-OUT g x* $-$ ($\sum^+$ *y*. *g* $(x, y) *$ *indicator* **E** $(y, x))$
(**is** - $=$ - $-$ *?g-in-E*) **unfolding** *d-OUT-def*
by(*subst nn-integral-diff*)(*simp-all add*: *AE-count-space finite2 split*: *split-indicator*)
also have *d-IN f x* $+$ (*d-OUT g x* $-$ ($\sum^+$ *y*. *g* $(x, y) *$ *indicator* **E** $(y, x)))$ $-$ *?g-in* $=$
((*d-IN f x* $+$ *d-OUT g x*) $-$ ($\sum^+$ *y*. *g* $(x, y) *$ *indicator* **E** $(y, x)))$ $-$ *?g-in*
by (*subst add-diff-eq-ennreal*) (*auto simp*: *d-OUT-def intro*!: *nn-integral-mono split*: *split-indicator*)
also have *d-OUT g x* $=$ *d-IN g x* **using** *x g* **by**(*auto dest*: *flowD-KIR*)
also have ... $=$ ($\sum^+$ *y*$\in$*UNIV*. *g* $(y, x) *$ *indicator* $(-$ **E**$)$ $(y, x))$ $+$ ($\sum^+$ *y*. *g* $(y, x) *$ *indicator* **E** $(y, x))$
(**is** - $=$ *?x* $+$ *?g-in-E'*)
by(*subst nn-integral-add*[*symmetric*])(*auto intro*!: *nn-integral-cong simp add*: *d-IN-def AE-count-space split*: *split-indicator*)
also have *?x* $=$ *?g-in*
proof $-$
have *?x* $=$ ($\sum^+$ *y*. *g* $(y, x) *$ *indicator* $(-$ **E**$)$ $(x, y) *$ *indicator* $(-$ **E**$)$ $(y, x))$ $+$ *?g-in*
by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *AE-count-space dest*: *wf-residual-networkD*[*OF wf*] *split*: *split-indicator intro*!: *nn-integral-cong*)
also have ($\sum^+$ *y*. *g* $(y, x) *$ *indicator* $(-$ **E**$)$ $(x, y) *$ *indicator* $(-$ **E**$)$ $(y, x))$ $=$ *0*
using *RES.flowD-outside*[*OF g*]
by(*auto simp add*: *nn-integral-0-iff-AE AE-count-space split*: *split-indicator*)
finally show *?thesis* **by** *simp*
qed
also have (*d-IN f x* $+$ (*?g-in* $+$ *?g-in-E'*) $-$ *?g-in-E*) $-$ *?g-in* $=$
d-IN f x $+$ *?g-in-E'* $+$ *?g-in* $-$ *?g-in* $-$ *?g-in-E*
by (*subst diff-diff-commute-ennreal*) (*simp add*: *ac-simps*)
also have ... $=$ *d-IN f x* $+$ *?g-in-E'* $-$ *?g-in-E*
by (*subst ennreal-add-diff-cancel-right*) (*simp-all add*: *finite1*)
also have ... $=$ *d-IN* $(f \oplus g)$ *x*
using *flowD-outside*[*OF f*] *g-le-f RES.flowD-finite-OUT*[*OF g*, *of x*]
by(*rule IN-plus-flow*[*symmetric*])(*simp-all add*: *x*)
finally show *KIR* $(f \oplus g)$ *x* **by** *simp*
qed

definition *minus-flow* :: *'v flow* ⇒ *'v flow* ⇒ *'v flow* (**infixl** ‹⊖› *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*)
by(*simp add*: *minus-flow-def*)

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
  show *?f e* $\leq$ *capacity ?*Δ *e* **for** *e*
    using *value-le f-finite flowD-capacity*[*OF g*] *flowD-capacity*[*OF f*]
    by(*cases e*)(*auto simp add*: *source-in source-out value-flow*[*OF f source-out*] *value-flow*[*OF g source-out*] *less-top*
      intro!: *diff-le-self-ennreal diff-eq-0-ennreal ennreal-minus-mono*)

  fix *x*
  assume $x \neq$ *source ?*Δ $x \neq$ *sink ?*Δ
  hence *x*: $x \neq$ *source* Δ $x \neq$ *sink* Δ **by** *simp-all*

  have *finite-f-in*: ($\sum^+$ *y*. *f* (*y*, *x*) ∗ *indicator A y*) $\neq$ *top* **for** *A*
    using *flowD-finite-IN*[*OF f*, *of x*]
    by(*rule neq-top-trans*)(*auto simp add*: *x d-IN-def split*: *split-indicator intro*!: *nn-integral-mono*)
  have *finite-f-out*: ($\sum^+$ *y*. *f* (*x*, *y*) ∗ *indicator A y*) $\neq$ *top* **for** *A*
    using *flowD-finite-OUT*[*OF f*, *of x*]
    by(*rule neq-top-trans*)(*auto simp add*: *x d-OUT-def split*: *split-indicator intro*!: *nn-integral-mono*)
  have *finite-f*[*simp*]: *f* (*x*, *y*) $\neq$ *top* *f* (*y*, *x*) $\neq$ *top* **for** *y*
    using *finite-f-in*[*of* {*y*}] *finite-f-out*[*of* {*y*}] **by** *auto*

  have *finite-g-in*: ($\sum^+$ *y*. *g* (*y*, *x*) ∗ *indicator A y*) $\neq$ *top* **for** *A*
    using *flowD-finite-IN*[*OF g*, *of x*]
    by(*rule neq-top-trans*)(*auto simp add*: *x d-IN-def split*: *split-indicator intro*!: *nn-integral-mono*)
  have *finite-g-out*: ($\sum^+$ *y*. *g* (*x*, *y*) ∗ *indicator A y*) $\neq$ *top* **for** *A*
    using *flowD-finite-OUT*[*OF g x*]
    by(*rule neq-top-trans*)(*auto simp add*: *x d-OUT-def split*: *split-indicator intro*!: *nn-integral-mono*)
  have *finite-g*[*simp*]: *g* (*x*, *y*) $\neq$ *top* *g* (*y*, *x*) $\neq$ *top* **for** *y*

using *finite-g-in*[*of* {*y*}] *finite-g-out*[*of* {*y*}] **by** *auto*

have *d-OUT* $(f \ominus g)$ $x = (\sum^+ y.\ (f\ (x,\ y) - g\ (x,\ y)) * $ *indicator* $\mathbf{E}$ $(x,\ y) *$ *indicator* $\{y.\ g\ (x,\ y) \leq f\ (x,\ y)\}\ y) +$
$(\sum^+ y.\ (g\ (y,\ x) - f\ (y,\ x)) *$ *indicator* $\mathbf{E}$ $(y,\ x) *$ *indicator* $\{y.\ f\ (y,\ x) < g\ (y,\ x)\}\ y)$
(**is** - = *?out* + *?in* **is** - = $(\sum^+ y{\in}$-. - $*$ *?f-ge-g y*) + $(\sum^+ y{\in}$-. - $*$ *?g-gt-f y*))
using *flowD-finite*[*OF g*]
apply(*subst nn-integral-add*[*symmetric*])
apply(*auto 4 4 simp add*: *d-OUT-def not-le less-top*[*symmetric*] *intro*!: *nn-integral-cong dest*!: *wf-residual-networkD*[*OF wf*] *split*: *split-indicator intro*!: *diff-eq-0-ennreal*)
done
also have *?in* = $(\sum^+ y.\ (g\ (y,\ x) - f\ (y,\ x)) *$ *?g-gt-f y*)
using *flowD-outside*[*OF f*] *flowD-outside*[*OF g*] **by**(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = $(\sum^+ y{\in}UNIV.\ g\ (y,\ x) *$ *?g-gt-f y*) $-$ $(\sum^+ y.\ f\ (y,\ x) *$ *?g-gt-f y*) (**is** - = *?g-in* $-$ *?f-in*)
using *finite-f-in*
by(*subst nn-integral-diff*[*symmetric*])(*auto simp add*: *AE-count-space split*: *split-indicator intro*!: *nn-integral-cong*)
also have *?out* = $(\sum^+ y.\ (f\ (x,\ y) - g\ (x,\ y)) *$ *?f-ge-g y*)
using *flowD-outside*[*OF f*] *flowD-outside*[*OF g*] **by**(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = $(\sum^+ y.\ f\ (x,\ y) *$ *?f-ge-g y*) $-$ $(\sum^+ y.\ g\ (x,\ y) *$ *?f-ge-g y*) (**is** - = *?f-out* $-$ *?g-out*)
using *finite-g-out*
by(*subst nn-integral-diff*[*symmetric*])(*auto simp add*: *AE-count-space split*: *split-indicator intro*!: *nn-integral-cong*)
also have *?f-out* = *d-OUT f x* $-$ $(\sum^+ y.\ f\ (x,\ y) *$ *indicator* $\{y.\ f\ (x,\ y) < g\ (x,\ y)\}\ y)$ (**is** - = - $-$ *?f-out-less*)
unfolding *d-OUT-def* **using** *flowD-finite*[*OF f*] **using** *finite-f-out*
by(*subst nn-integral-diff*[*symmetric*])(*auto split*: *split-indicator intro*!: *nn-integral-cong*)
also have *?g-out* = *d-OUT g x* $-$ $(\sum^+ y.\ g\ (x,\ y) *$ *indicator* $\{y.\ f\ (x,\ y) < g\ (x,\ y)\}\ y)$ (**is** - = - $-$ *?g-less-f*)
unfolding *d-OUT-def* **using** *flowD-finite*[*OF g*] *finite-g-out*
by(*subst nn-integral-diff*[*symmetric*])(*auto split*: *split-indicator intro*!: *nn-integral-cong*)
also have *d-OUT f x* $-$ *?f-out-less* $-$ (*d-OUT g x* $-$ *?g-less-f*) + (*?g-in* $-$ *?f-in*) =
(*?g-less-f* + (*d-OUT f x* $-$ *?f-out-less*)) $-$ *d-OUT g x* + (*?g-in* $-$ *?f-in*)
by (*subst diff-diff-ennreal′*)
(*auto simp*: *ac-simps d-OUT-def nn-integral-diff*[*symmetric*] *finite-g-out finite-f-out intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... = *?g-less-f* + *d-OUT f x* $-$ *?f-out-less* $-$ *d-OUT g x* + (*?g-in* $-$ *?f-in*)
by (*subst add-diff-eq-ennreal*)
(*auto simp*: *d-OUT-def intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... = *d-OUT f x* + *?g-less-f* $-$ *?f-out-less* $-$ *d-OUT g x* + (*?g-in* $-$ *?f-in*)
by (*simp add*: *ac-simps*)

also have ... = *d-OUT f x* + (*?g-less-f* − *?f-out-less*) − *d-OUT g x* + (*?g-in* − *?f-in*)
by (*subst add-diff-eq-ennreal*[*symmetric*])
(*auto intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... = (*?g-in* − *?f-in*) + ((*?g-less-f* − *?f-out-less*) + *d-OUT f x* − *d-OUT g x*)
by (*simp add*: *ac-simps*)
also have ... = ((*?g-in* − *?f-in*) + ((*?g-less-f* − *?f-out-less*) + *d-OUT f x*)) − *d-OUT g x*
apply (*subst add-diff-eq-ennreal*)
apply (*simp-all add*: *d-OUT-def*)
apply (*subst nn-integral-diff*[*symmetric*])
apply (*auto simp*: *AE-count-space finite-f-out nn-integral-add*[*symmetric*] *not-less diff-add-cancel-ennreal intro*!: *nn-integral-mono split*: *split-indicator*)
done
also have ... = ((*?g-less-f* − *?f-out-less*) + (*d-OUT f x* + (*?g-in* − *?f-in*))) − *d-OUT g x*
by (*simp add*: *ac-simps*)
also have ... = ((*?g-less-f* − *?f-out-less*) + (*d-IN f x* + (*?g-in* − *?f-in*))) − *d-IN g x*
unfolding *flowD-KIR*[*OF f x*] *flowD-KIR*[*OF g x*] **..**
also have ... = (*?g-less-f* − *?f-out-less*) + ((*d-IN f x* + (*?g-in* − *?f-in*)) − *d-IN g x*)
apply (*subst* (*2*) *add-diff-eq-ennreal*)
apply (*simp-all add*: *d-IN-def*)
apply (*subst nn-integral-diff*[*symmetric*])
apply (*auto simp*: *AE-count-space finite-f-in finite-f-out nn-integral-add*[*symmetric*] *not-less ennreal-ineq-diff-add*[*symmetric*]
intro!: *nn-integral-mono split*: *split-indicator*)
done
also have ... = (*?g-less-f* − *?f-out-less*) + (*d-IN f x* + *?g-in* − *d-IN g x* − *?f-in*)
by (*subst* (*2*) *add-diff-eq-ennreal*) (*auto intro*!: *nn-integral-mono split*: *split-indicator simp*: *diff-diff-commute-ennreal*)
also have ... = (*?g-less-f* − *?f-out-less*) + (*d-IN f x* − (*d-IN g x* − *?g-in*) − *?f-in*)
apply (*subst diff-diff-ennreal'*)
apply (*auto simp*: *d-IN-def intro*!: *nn-integral-mono split*: *split-indicator*)
apply (*subst nn-integral-diff*[*symmetric*])
apply (*auto simp*: *AE-count-space finite-g-in intro*!: *nn-integral-mono split*: *split-indicator*)
done
also have ... =(*d-IN f x* − *?f-in*) − (*d-IN g x* − *?g-in*) + (*?g-less-f* − *?f-out-less*)
by (*simp add*: *ac-simps diff-diff-commute-ennreal*)
also have *?g-less-f* − *?f-out-less* = ($\sum^+$ *y*. (*g* (*x*, *y*) − *f* (*x*, *y*)) ∗ *indicator* {*y*. *f* (*x*, *y*) < *g* (*x*, *y*)} *y*) **using** *finite-f-out*
by(*subst nn-integral-diff*[*symmetric*])(*auto simp add*: *AE-count-space split*: *split-indicator intro*!: *nn-integral-cong*)
also have ... = ($\sum^+$ *y*. (*g* (*x*, *y*) − *f* (*x*, *y*)) ∗ *indicator* **E** (*x*, *y*) ∗ *indicator*

$\{y.\ f\ (x,\ y) < g\ (x,\ y)\}\ y)$ (**is** - = *?diff-out*)
using *flowD-outside*[*OF f*] *flowD-outside*[*OF g*] **by**(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have *d-IN f x* − *?f-in* = ($\sum^+$ *y*. *f* (*y*, *x*) ∗ *indicator* {*y*. *g* (*y*, *x*) ≤ *f* (*y*, *x*)} *y*)
unfolding *d-IN-def* **using** *finite-f-in*
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space split*: *split-indicator intro*!: *nn-integral-cong*)
done
also have *d-IN g x* − *?g-in* = ($\sum^+$ *y*. *g* (*y*, *x*) ∗ *indicator* {*y*. *g* (*y*, *x*) ≤ *f* (*y*, *x*)} *y*)
unfolding *d-IN-def* **using** *finite-g-in*
by(*subst nn-integral-diff*[*symmetric*])(*auto simp add*: *flowD-finite*[*OF g*] *AE-count-space split*: *split-indicator intro*!: *nn-integral-cong*)
also have ($\sum^+$ *y*∈*UNIV*. *f* (*y*, *x*) ∗ *indicator* {*y*. *g* (*y*, *x*) ≤ *f* (*y*, *x*)} *y*) − . . .
= ($\sum^+$ *y*. (*f* (*y*, *x*) − *g* (*y*, *x*)) ∗ *indicator* {*y*. *g* (*y*, *x*) ≤ *f* (*y*, *x*)} *y*)
using *finite-g-in*
by(*subst nn-integral-diff*[*symmetric*])(*auto simp add*: *flowD-finite*[*OF g*] *AE-count-space split*: *split-indicator intro*!: *nn-integral-cong*)
also have . . . = ($\sum^+$ *y*. (*f* (*y*, *x*) − *g* (*y*, *x*)) ∗ *indicator* **E** (*y*, *x*) ∗ *indicator* {*y*. *g* (*y*, *x*) ≤ *f* (*y*, *x*)} *y*)
using *flowD-outside*[*OF f*] *flowD-outside*[*OF g*] **by**(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have . . . + *?diff-out* = *d-IN ?f x*
using *flowD-finite*[*OF g*]
apply(*subst nn-integral-add*[*symmetric*])
apply(*auto 4 4 simp add*: *d-IN-def not-le less-top*[*symmetric*] *intro*!: *nn-integral-cong dest*!: *wf-residual-networkD*[*OF wf*] *split*: *split-indicator intro*: *diff-eq-0-ennreal*)
done
finally show *KIR ?f x* **.**
qed

lemma *value-minus-flow*:
assumes *f*: *flow* Δ *f*
and *g*: *flow* Δ *g*
and *value-le*: *value-flow* Δ *g* ≤ *value-flow* Δ *f*
and *source-out*: ⋀*y*. *edge* Δ (*source* Δ) *y* ⟷ *y* = *x*
shows *value-flow* Δ (*f* ⊖ *g*) = *value-flow* Δ *f* − *value-flow* Δ *g* (**is** *?value*)
proof −
have *value-flow* Δ (*f* ⊖ *g*) = ($\sum^+$ *y*∈**OUT** (*source* Δ). (*f* ⊖ *g*) (*source* Δ, *y*))
by(*subst d-OUT-alt-def*)(*auto simp add*: *flowD-outside*[*OF f*] *flowD-outside*[*OF g*] *source-in*)
also have . . . = ($\sum^+$ *y*. (*f* (*source* Δ, *y*) − *g* (*source* Δ, *y*)) ∗ *indicator* {*x*} *y*)
by(*subst nn-integral-count-space-indicator*)(*auto intro*!: *nn-integral-cong split*: *split-indicator simp add*: *outgoing-def source-out*)
also have . . . = *f* (*source* Δ, *x*) − *g* (*source* Δ, *x*)
using *value-le value-flow*[*OF f source-out*] *value-flow*[*OF g source-out*]
by(*auto simp add*: *one-ennreal-def*[*symmetric*] *max-def not-le intro*: *antisym*)

```
  also have ... = value-flow Δ f − value-flow Δ g using f g source-out by(simp
add: value-flow)
  finally show ?value .
qed

context
  fixes α
  defines α ≡ (SUP g∈{g. flow Δ g}. value-flow Δ g)
begin

lemma flow-by-value:
  assumes v < α
  and real[rule-format]: ∀f. α = ⊤ ⟶ flow Δ f ⟶ value-flow Δ f < α
  obtains f where flow Δ f value-flow Δ f = v
proof −
  have α-pos: α > 0 using assms by (auto simp add: zero-less-iff-neq-zero)
  from ‹v < α› obtain f where f: flow Δ f and v: v < value-flow Δ f
    unfolding α-def less-SUP-iff by blast
  have [simp]: value-flow Δ f ≠ ⊤
  proof
    assume val: value-flow Δ f = ⊤
   from f have value-flow Δ f ≤ α unfolding α-def by(blast intro: SUP-upper2)
    with val have α = ⊤ by (simp add: top-unique)
    from real[OF this f] val show False by simp
  qed
  let ?f = λe. (v / value-flow Δ f) ∗ f e
  note f
  moreover
  have ∗: 0 < value-flow Δ f
    using ‹v < value-flow Δ f› by (auto simp add: zero-less-iff-neq-zero)
  then have v / value-flow Δ f ≤ 1 using v
    by (auto intro!: divide-le-posI-ennreal)
  ultimately have flow Δ ?f by (rule scale-flow)
  moreover {
    have value-flow Δ ?f = v ∗ (value-flow Δ f / value-flow Δ f)
      by(subst value-scale-flow)(simp add: divide-ennreal-def ac-simps)
   also have ... = v using ∗ by(subst ennreal-divide-self) (auto simp: less-top[symmetric])
    also note calculation }
  ultimately show ?thesis by(rule that)
qed

theorem ex-max-flow':
  assumes wf: wf-residual-network
  assumes source-out: ⋀y. edge Δ (source Δ) y ⟷ y = x
  and nontrivial: V − {source Δ, sink Δ} ≠ {}
  and real: α = ennreal α' and α'-nonneg[simp]: 0 ≤ α'
  shows ∃f. flow Δ f ∧ value-flow Δ f = α ∧ (∀x. d-IN f x ≤ value-flow Δ f)
proof −
  have α'-not-neg[simp]: ¬ α' < 0
```

using α'-*nonneg* **by** *linarith*

let *?v* = λ*i*. (*1* − (*1* / *2*) ^ *i*) ∗ α
let *?v-r* = λ*i*. *ennreal* ((*1* − (*1* / *2*) ^ *i*) ∗ α')
have *v-eq*: *?v i* = *?v-r i* **for** *i*
by (*auto simp*: *real ennreal-mult power-le-one ennreal-lessI ennreal-minus*[*symmetric*] *ennreal-power*[*symmetric*] *divide-ennreal-def*)
have ∃*f*. *flow* Δ *f* ∧ *value-flow* Δ *f* = *?v i* **for** *i* :: *nat*
proof(*cases* α = *0*)
case *True* **thus** *?thesis* **by**(*auto intro*!: *exI*[**where** *x*=λ-. *0*])
next
case *False*
then have *?v i* < α
unfolding *v-eq* **by** (*auto simp*: *real field-simps intro*!: *ennreal-lessI*) (*simp-all add*: *less-le*)
then obtain *f* **where** *flow* Δ *f* **and** *value-flow* Δ *f* = *?v i*
by(*rule flow-by-value*)(*simp add*: *real*)
thus *?thesis* **by** *blast*
qed
then obtain *f-aux* **where** *f-aux*: ⋀*i*. *flow* Δ (*f-aux i*)
and *value-aux*: ⋀*i*. *value-flow* Δ (*f-aux i*) = *?v-r i*
unfolding *v-eq* **by** *moura*

define *f-i* **where** *f-i* = *rec-nat* (λ-. *0*) (λ*i f-i*.
let g = *f-aux* (*Suc i*) ⊖ *f-i*;
k-i = *SOME k*. *k* ≤ *g* ∧ *flow* (*residual-network f-i*) *k* ∧ *value-flow* (*residual-network f-i*) *k* = *value-flow* (*residual-network f-i*) *g* ∧ (∀ *x*. *d-IN k x* ≤ *value-flow* (*residual-network f-i*) *k*)
in f-i ⊕ *k-i*)
let *?P* = λ*i k*. *k* ≤ *f-aux* (*Suc i*) ⊖ *f-i i* ∧ *flow* (*residual-network* (*f-i i*)) *k* ∧ *value-flow* (*residual-network* (*f-i i*)) *k* = *value-flow* (*residual-network* (*f-i i*)) (*f-aux* (*Suc i*) ⊖ *f-i i*) ∧ (∀ *x*. *d-IN k x* ≤ *value-flow* (*residual-network* (*f-i i*)) *k*)
define *k-i* **where** *k-i i* = *Eps* (*?P i*) **for** *i*

have *f-i-simps* [*simp*]: *f-i 0* = (λ-. *0*) *f-i* (*Suc i*) = *f-i i* ⊕ *k-i i* **for** *i*
by(*simp-all add*: *f-i-def Let-def k-i-def*)

have *k-i*: *flow* (*residual-network* (*f-i i*)) (*k-i i*) (**is** *?k-i*)
and *value-k-i*: *value-flow* (*residual-network* (*f-i i*)) (*k-i i*) = *value-flow* (*residual-network* (*f-i i*)) (*f-aux* (*Suc i*) ⊖ *f-i i*) (**is** *?value-k-i*)
and *IN-k-i*: *d-IN* (*k-i i*) *x* ≤ *value-flow* (*residual-network* (*f-i i*)) (*k-i i*) (**is** *?IN-k-i*)
and *value-diff*: *value-flow* (*residual-network* (*f-i i*)) (*f-aux* (*Suc i*) ⊖ *f-i i*) = *value-flow* Δ (*f-aux* (*Suc i*)) − *value-flow* Δ (*f-i i*) (**is** *?value-diff*)
if *flow-network* Δ (*f-i i*) **and** *value-f-i*: *value-flow* Δ (*f-i i*) = *value-flow* Δ (*f-aux i*) **for** *i x*
proof −
let *?RES* = *residual-network* (*f-i i*)
interpret *fn*: *flow-network* Δ *f-i i* **by**(*rule that*)

interpret *RES*: *flow-attainability ?RES* **using** *wf fn.g* **by**(*rule residual-flow-attainability*)
have *le*: *value-flow* Δ (*f-i i*) $\leq$ *value-flow* Δ (*f-aux* (*Suc i*))
unfolding *value-aux value-f-i*
unfolding *v-eq* **by** (*rule ennreal-leI*) (*auto simp*: *field-simps*)
with *wf f-aux fn.g* **have** *res-flow*: *flow ?RES* (*f-aux* (*Suc i*) $\ominus$ *f-i i*)
using *flowD-finite*[*OF f-aux*] *source-out*
by(*rule minus-flow*)
show *value'*: *?value-diff* **by**(*simp add*: *value-minus-flow*[*OF f-aux fn.g le source-out*])
also have ... $< \top$
unfolding *value-aux v-eq* **by** (*auto simp*: *less-top*[*symmetric*])
finally have *value-flow ?RES* (*f-aux* (*Suc i*) $\ominus$ *f-i i*) $\neq \top$ **by** *simp*
then have *fn'*: *flow-network ?RES* (*f-aux* (*Suc i*) $\ominus$ *f-i i*)
using *nontrivial res-flow* **by**(*unfold-locales*) *simp-all*
then interpret *fn'*: *flow-network ?RES f-aux* (*Suc i*) $\ominus$ *f-i i* **.**
from *fn'.flow-cleanup* **show** *?k-i ?value-k-i ?IN-k-i* **unfolding** *k-i-def* **by**(*rule someI2-ex*; *blast*)+
qed

have *fn-i*: *flow-network* Δ (*f-i i*)
and *value-f-i*: *value-flow* Δ (*f-i i*) = *value-flow* Δ (*f-aux i*)
and *d-IN-i*: *d-IN* (*f-i i*) *x* $\leq$ *value-flow* Δ (*f-i i*) **for** *i x*
proof(*induction i*)
case *0*
{ case *1* **show** *?case* **using** *nontrivial* **by**(*unfold-locales*)(*simp-all add*: *f-aux value-aux*) **}**
{ case *2* **show** *?case* **by**(*simp add*: *value-aux*) **}**
{ case *3* **show** *?case* **by**(*simp*) **}**
next
case (*Suc i*)
interpret *fn*: *flow-network* Δ *f-i i* **using** *Suc.IH*(*1*) **.**
let *?RES* = *residual-network* (*f-i i*)

have *k-i*: *flow ?RES* (*k-i i*)
and *value-k-i*: *value-flow ?RES* (*k-i i*) = *value-flow ?RES* (*f-aux* (*Suc i*) $\ominus$ *f-i i*)
and *d-IN-k-i*: *d-IN* (*k-i i*) *x* $\leq$ *value-flow ?RES* (*k-i i*) **for** *x*
using *Suc.IH*(*1*$-$*2*) **by**(*rule k-i value-k-i IN-k-i*)+

interpret *RES*: *flow-attainability ?RES* **using** *wf fn.g* **by**(*rule residual-flow-attainability*)
have *le*: *value-flow* Δ (*f-i i*) $\leq$ *value-flow* Δ (*f-aux* (*Suc i*))
unfolding *value-aux Suc.IH*(*2*) *v-eq* **using** α'*-nonneg* **by**(*intro ennreal-leI*)(*simp add*: *real field-simps*)
{ case *1* **show** *?case* **unfolding** *f-i-simps*
proof
show *flow* Δ (*f-i i* $\oplus$ *k-i i*) **using** *wf fn.g k-i* **by**(*rule flow-residual-add*)
with *RES.flowD-finite*[*OF k-i*] **show** *value-flow* Δ (*f-i i* $\oplus$ *k-i i*) $\neq \top$
by(*simp add*: *value-flow*[*OF - source-out*])
qed(*rule nontrivial*) **}**
from *value-k-i* **have** *value-k*: *value-flow ?RES* (*k-i i*) = *value-flow* Δ (*f-aux*

$(Suc\ i)) - value\text{-}flow\ \Delta\ (f\text{-}aux\ i)$
by(*simp add*: *value-minus-flow*[*OF f-aux fn.g le source-out*] *Suc.IH*)
{ case *2* **show** *?case* **using** *value-k*
by(*auto simp add*: *source-out value-plus-flow*[*OF wf fn.g k-i*] *Suc.IH value-aux field-simps intro*!: *ennreal-leI*) **}**
note *value-f* = *this*
{ case *3*
have $d\text{-}IN\ (f\text{-}i\ i \oplus k\text{-}i\ i)\ x \leq d\text{-}IN\ (f\text{-}i\ i)\ x + d\text{-}IN\ (k\text{-}i\ i)\ x$
using *fn.g k-i* **by**(*rule d-IN-plus-flow*[*OF wf*])
also have ... $\leq value\text{-}flow\ \Delta\ (f\text{-}i\ i) + d\text{-}IN\ (k\text{-}i\ i)\ x$ **using** *Suc.IH*(*3*) **by**(*rule add-right-mono*)
also have ... $\leq value\text{-}flow\ \Delta\ (f\text{-}i\ i) + value\text{-}flow\ ?RES\ (k\text{-}i\ i)$ **using** *d-IN-k-i* **by**(*rule add-left-mono*)
also have ... $= value\text{-}flow\ \Delta\ (f\text{-}i\ (Suc\ i))$ **unfolding** *value-f Suc.IH*(*2*) *value-k*
by(*auto simp add*: *value-aux field-simps intro*!: *ennreal-leI*)
finally show *?case* **by** *simp* **}**
qed
interpret *fn*: *flow-network* Δ *f-i i* **for** *i* **by**(*rule fn-i*)
note *k-i* = *k-i*[*OF fn-i value-f-i*] **and** *value-k-i* = *value-k-i*[*OF fn-i value-f-i*]
and *IN-k-i* = *IN-k-i*[*OF fn-i value-f-i*] **and** *value-diff* = *value-diff*[*OF fn-i value-f-i*]

have $\exists x \geq 0.\ f\text{-}i\ i\ e = ennreal\ x$ **for** *i e*
using *fn.finite-g*[*of i e*] **by** (*cases f-i i e*) *auto*
then obtain *f-i'* **where** *f-i'*: $\bigwedge i\ e.\ f\text{-}i\ i\ e = ennreal\ (f\text{-}i'\ i\ e)$ **and** [*simp*]: $\bigwedge i\ e.\ 0 \leq f\text{-}i'\ i\ e$
by *metis*

{ fix *i e*
obtain *x y* :: *'v* **where** *e*: $e = (x, y)$ **by**(*cases e*)
have $k\text{-}i\ i\ (x, y) \leq d\text{-}IN\ (k\text{-}i\ i)\ y$ **by** (*rule d-IN-ge-point*)
also have ... $\leq value\text{-}flow\ (residual\text{-}network\ (f\text{-}i\ i))\ (k\text{-}i\ i)$ **by**(*rule IN-k-i*)
also have ... $< \top$ **using** *value-k-i*[*of i*] *value-diff*[*of i*]
by(*simp add*: *value-k-i value-f-i value-aux real less-top*[*symmetric*])
finally have $\exists x \geq 0.\ k\text{-}i\ i\ e = ennreal\ x$
by(*cases k-i i e*)(*auto simp add*: *e*) **}**
then obtain *k-i'* **where** *k-i'*: $\bigwedge i\ e.\ k\text{-}i\ i\ e = ennreal\ (k\text{-}i'\ i\ e)$ **and** *k-i'-nonneg*[*simp*]: $\bigwedge i\ e.\ 0 \leq k\text{-}i'\ i\ e$
by *metis*

have *wf-k*: $(x, y) \in \mathbf{E} \Longrightarrow k\text{-}i\ i\ (y, x) \leq f\text{-}i\ i\ (x, y) + k\text{-}i\ i\ (x, y)$ **for** *i x y*
using *flowD-capacity*[*OF k-i*, *of i* (*y*, *x*)]
by (*auto split*: *if-split-asm dest*: *wf-residual-networkD*[*OF wf*] *elim*: *order-trans*)

have *f-i'-0*[*simp*]: $f\text{-}i'\ 0 = (\lambda\text{-}.\ 0)$ **using** *f-i-simps*(*1*) **by** (*simp del*: *f-i-simps add*: *fun-eq-iff f-i'*)

have *f-i'-Suc*[*simp*]: $f\text{-}i'\ (Suc\ i)\ e = (if\ e \in \mathbf{E}\ then\ f\text{-}i'\ i\ e + (k\text{-}i'\ i\ e - k\text{-}i'\ i$

```
(prod.swap e)) else 0) for i e
    using f-i-simps(2)[of i, unfolded fun-eq-iff, THEN spec, of e] wf-k[of fst e snd
e i]
    by (auto simp del: f-i-simps ennreal-plus
            simp add: fun-eq-iff f-i' k-i' ennreal-plus[symmetric] ennreal-minus split:
if-split-asm)

  have k-i'-le: k-i' i e ≤ α' / 2 ^ (Suc i) for i e
  proof −
    obtain x y where e: e = (x, y) by(cases e)
    have k-i' i (x, y) ≤ d-IN (k-i' i) y by (rule d-IN-ge-point)
    also have ... ≤ value-flow (residual-network (f-i i)) (k-i' i)
      using IN-k-i[of i y] by(simp add: k-i'[abs-def])
    also have ... = α' / 2 ^ (Suc i) using value-k-i[of i] value-diff[of i]
       by(simp add: value-f-i value-aux real k-i'[abs-def] field-simps ennreal-minus
mult-le-cancel-left1)
    finally show ?thesis using e by simp
  qed

  have convergent: convergent (λi. f-i' i e) for e
  proof(cases α' > 0)
    case False
    obtain x y where [simp]: e = (x, y) by(cases e)

    { fix i
      from False α'-nonneg have α' = 0 by simp
      moreover have f-i i (x, y) ≤ d-IN (f-i i) y by (rule d-IN-ge-point)
      ultimately have f-i i (x, y) = 0 using d-IN-i[of i y]
        by(simp add: value-f-i value-aux real) }
    thus ?thesis by(simp add: f-i' convergent-const)
  next
    case α'-pos: True
    show ?thesis
    proof(rule real-Cauchy-convergent Cauchy-real-Suc-diff)+
      fix n
      have |k-i' n e − k-i' n (prod.swap e)| ≤ |k-i' n e| + |k-i' n (prod.swap e)|
        by (rule abs-triangle-ineq4)
      then have |k-i' n e − k-i' n (prod.swap e)| ≤ α' / 2 ^ n
        using k-i'-le[of n e] k-i'-le[of n prod.swap e] by simp
      then have |f-i' (Suc n) e − f-i' n e| ≤ α' / 2 ^ n
        using flowD-outside[OF fn.g] by (cases e) (auto simp: f-i')
      thus |f-i' (Suc n) e − f-i' n e| ≤ α' / 2 ^ n by simp
    qed simp
  qed
 then obtain f' where f': ⋀e. (λi. f-i' i e) ⟶ f' e unfolding convergent-def
by metis
  hence f: ⋀e. (λi. f-i i e) ⟶ ennreal (f' e) unfolding f-i' by simp

  have f'-nonneg: 0 ≤ f' e for e
```

by (*rule LIMSEQ-le-const*[*OF f'*]) *auto*

let *?k* = λ*i x y*. (*k-i' i* (*x*, *y*) − *k-i' i* (*y*, *x*)) ∗ *indicator* **E** (*x*, *y*)
have *sum-i'*: *f-i' i* (*x*, *y*) = ($\sum$*j*<*i*. *?k j x y*) **for** *x y i*
by (*induction i*) *auto*

have *summable-nk*: *summable* (λ*i*. |*?k i x y*|) **for** *x y*
proof(*rule summable-rabs-comparison-test*)
show ∃ *N*. ∀ *i*≥*N*. |*?k i x y*| ≤ α' ∗ (*1* / *2*) ^ *i*
proof (*intro exI allI impI*)
fix *i* **have** |*?k i x y*| ≤ *k-i' i* (*x*, *y*) + *k-i' i* (*y*, *x*)
by (*auto intro*!: *abs-triangle-ineq4* [*THEN order-trans*] *split*: *split-indicator*)
also have ... ≤ α' ∗ (*1* / *2*) ^ *i*
using *k-i'-le*[*of i* (*x*, *y*)] *k-i'-le*[*of i* (*y*, *x*)] α'*-nonneg k-i'-nonneg*[*of i* (*x*, *y*)] *k-i'-nonneg*[*of i* (*y*, *x*)]
by(*auto simp add*: *abs-real-def power-divide split*: *split-indicator*)
finally show |*?k i x y*| ≤ α' ∗ (*1* / *2*) ^ *i*
by *simp*
qed
show *summable* (λ*i*. α' ∗ (*1* / *2*) ^ *i*)
by(*rule summable-mult complete-algebra-summable-geometric*)+ *simp*
qed
hence *summable-k*: *summable* (λ*i*. *?k i x y*) **for** *x y* **by**(*auto intro*: *summable-norm-cancel*)

have *suminf*: ($\sum$ *i*. (*k-i' i* (*x*, *y*) − *k-i' i* (*y*, *x*)) ∗ *indicator* **E** (*x*, *y*)) = *f'* (*x*, *y*) **for** *x y*
by(*rule LIMSEQ-unique*[*OF summable-LIMSEQ*])(*simp-all add*: *sum-i'*[*symmetric*] *f' summable-k*)

have *flow*: *flow* Δ *f'*
proof
fix *e*
have *f' e* ≤ *Sup* {..*capacity* Δ *e*} **using** - *f*
by(*rule Sup-lim*)(*simp add*: *flowD-capacity*[*OF fn.g*])
then show *f' e* ≤ *capacity* Δ *e* **by** *simp*
next
fix *x*
assume *x*: *x* ≠ *source* Δ *x* ≠ *sink* Δ

have *integrable-f-i*: *integrable* (*count-space UNIV*) (λ*y*. *f-i' i* (*x*, *y*)) **for** *i*
using *flowD-finite-OUT*[*OF fn.g x*, *of i*] **by**(*auto intro*!: *integrableI-bounded simp add*: *f-i' d-OUT-def less-top*)
have *integrable-f-i'*: *integrable* (*count-space UNIV*) (λ*y*. *f-i' i* (*y*, *x*)) **for** *i*
using *flowD-finite-IN*[*OF fn.g*, *of x i*] *x* **by**(*auto intro*!: *integrableI-bounded simp add*: *f-i' d-IN-def less-top*)

have *integral-k-bounded*: ($\sum^+$ *y*. *norm* (*?k i x y*)) ≤ α' / *2* ^ *i* (**is** *?thesis1*)
and *integral-k'-bounded*: ($\sum^+$ *y*. *norm* (*?k i y x*)) ≤ α' / *2* ^ *i* (**is** *?thesis2*) **for** *i*

proof −
define *b* **where** *b* = ($\sum^+$ *y*. *k-i i* (*x*, *y*) + *k-i i* (*y*, *x*))
have *b* = *d-OUT* (*k-i i*) *x* + *d-IN* (*k-i i*) *x* **unfolding** *b-def*
by(*subst nn-integral-add*)(*simp-all add*: *d-OUT-def d-IN-def*)
also have *d-OUT* (*k-i i*) *x* = *d-IN* (*k-i i*) *x* **using** *k-i* **by**(*rule flowD-KIR*)(*simp-all add*: *x*)
also have ... + ... ≤ *value-flow* Δ (*k-i i*) + *value-flow* Δ (*k-i i*)
using *IN-k-i*[*of i x*, *simplified*] **by**−(*rule add-mono*)
also have ... ≤ α′ / *2* ^ *i* **using** *value-k-i*[*of i*] *value-diff*[*of i*]
by(*simp add*: *value-aux value-f-i field-simps ennreal-minus-if ennreal-plus-if mult-le-cancel-left1*
del: *ennreal-plus*)
also have ($\sum^+$ *y*∈*UNIV*. *norm* (*?k i x y*)) ≤ *b* **and** ($\sum^+$ *y*. *norm* (*?k i y x*)) ≤ *b* **unfolding** *b-def*
by(*rule nn-integral-mono*; *simp add*: *abs-real-def k-i′ ennreal-plus-if del*: *ennreal-plus split*: *split-indicator*)+
ultimately show *?thesis1 ?thesis2* **by**(*auto*)
qed

have *integrable-k*: *integrable* (*count-space UNIV*) (λ*y*. *?k i x y*)
and *integrable-k′*: *integrable* (*count-space UNIV*) (λ*y*. *?k i y x*) **for** *i*
using *integral-k-bounded*[*of i*] *integral-k′-bounded*[*of i*] *real*
by(*auto intro*!: *integrableI-bounded simp*: *less-top*[*symmetric*] *top-unique ennreal-divide-eq-top-iff*)

have *summable′-k*: *summable* (λ*i*. ∫ *y*. |*?k i x y*| ∂*count-space UNIV*)
proof(*rule summable-comparison-test*)
have |∫ *y*. |*?k i x y*| ∂*count-space UNIV*| ≤ α′ ∗ (*1* / *2*) ^ *i* **for** *i*
using *integral-norm-bound-ennreal*[*OF integrable-norm*, *OF integrable-k*, *of i*] *integral-k-bounded*[*of i*]
by(*bestsimp simp add*: *real power-divide dest*: *order-trans*)
thus ∃ *N*. ∀ *i*≥*N*. *norm* (∫ *y*. |*?k i x y*| ∂*count-space UNIV*) ≤ α′ ∗ (*1* / *2*) ^ *i*
by *auto*
show *summable* (λ*i*. α′ ∗ (*1* / *2*) ^ *i*)
by(*rule summable-mult complete-algebra-summable-geometric*)+ *simp*
qed
have *summable′-k′*: *summable* (λ*i*. ∫ *y*. |*?k i y x*| ∂*count-space UNIV*)
proof(*rule summable-comparison-test*)
have |∫ *y*. |*?k i y x*| ∂*count-space UNIV*| ≤ α′ ∗ (*1* / *2*) ^ *i* **for** *i*
using *integral-norm-bound-ennreal*[*OF integrable-norm*, *OF integrable-k′*, *of i*] *integral-k′-bounded*[*of i*]
by(*bestsimp simp add*: *real power-divide dest*: *order-trans*)
thus ∃ *N*. ∀ *i*≥*N*. *norm* (∫ *y*. |*?k i y x*| ∂*count-space UNIV*) ≤ α′ ∗ (*1* / *2*) ^ *i* **by** *auto*
show *summable* (λ*i*. α′ ∗ (*1* / *2*) ^ *i*)
by(*rule summable-mult complete-algebra-summable-geometric*)+ *simp*
qed

have $(\lambda i.\ \int y.\ ?k\ i\ x\ y\ \partial count\text{-}space\ UNIV)$ *sums* $\int y.\ (\sum i.\ ?k\ i\ x\ y)$ $\partial count\text{-}space\ UNIV$
using *integrable-k* **by**(*rule sums-integral*)(*simp-all add*: *summable-nk summable′-k*)
also have $\ldots = \int y.\ f'\ (x, y)\ \partial count\text{-}space\ UNIV$ **by**(*rule Bochner-Integration.integral-cong*[*OF refl*])(*rule suminf*)
finally have $(\lambda i.\ \sum j<i.\ \int y.\ ?k\ j\ x\ y\ \partial count\text{-}space\ UNIV) \longlongrightarrow \ldots$ **unfolding** *sums-def* **.**
also have $(\lambda i.\ \sum j<i.\ \int y.\ ?k\ j\ x\ y\ \partial count\text{-}space\ UNIV) = (\lambda i.\ \int y.\ f\text{-}i'\ i\ (x, y)\ \partial count\text{-}space\ UNIV)$
unfolding *sum-i′* **by**(*rule ext Bochner-Integration.integral-sum*[*symmetric*] *integrable-k*)+
finally have $(\lambda i.\ ennreal\ (\int y.\ f\text{-}i'\ i\ (x, y)\ \partial count\text{-}space\ UNIV)) \longlongrightarrow ennreal\ (\int y.\ f'\ (x, y)\ \partial count\text{-}space\ UNIV)$ **by** *simp*
also have $(\lambda i.\ ennreal\ (\int y.\ f\text{-}i'\ i\ (x, y)\ \partial count\text{-}space\ UNIV)) = (\lambda i.\ d\text{-}OUT\ (f\text{-}i\ i)\ x)$
unfolding *d-OUT-def f-i′* **by**(*rule ext nn-integral-eq-integral*[*symmetric*] *integrable-f-i*)+ *simp*
also have $ennreal\ (\int y.\ f'\ (x, y)\ \partial count\text{-}space\ UNIV) = d\text{-}OUT\ f'\ x$
unfolding *d-OUT-def* **by**(*rule nn-integral-eq-integral*[*symmetric*])(*simp-all add*: *f′-nonneg*, *simp add*: *suminf*[*symmetric*] *integrable-suminf integrable-k summable-nk summable′-k*)
also have $(\lambda i.\ d\text{-}OUT\ (f\text{-}i\ i)\ x) = (\lambda i.\ d\text{-}IN\ (f\text{-}i\ i)\ x)$
using *flowD-KIR*[*OF fn.g x*] **by**(*simp*)
finally have $*$: $(\lambda i.\ d\text{-}IN\ (f\text{-}i\ i)\ x) \longlongrightarrow d\text{-}OUT\ (\lambda x.\ ennreal\ (f'\ x))\ x$ **.**

have $(\lambda i.\ \int y.\ ?k\ i\ y\ x\ \partial count\text{-}space\ UNIV)$ *sums* $\int y.\ (\sum i.\ ?k\ i\ y\ x)$ $\partial count\text{-}space\ UNIV$
using *integrable-k′* **by**(*rule sums-integral*)(*simp-all add*: *summable-nk summable′-k′*)
also have $\ldots = \int y.\ f'\ (y, x)\ \partial count\text{-}space\ UNIV$ **by**(*rule Bochner-Integration.integral-cong*[*OF refl*])(*rule suminf*)
finally have $(\lambda i.\ \sum j<i.\ \int y.\ ?k\ j\ y\ x\ \partial count\text{-}space\ UNIV) \longlongrightarrow \ldots$ **unfolding** *sums-def* **.**
also have $(\lambda i.\ \sum j<i.\ \int y.\ ?k\ j\ y\ x\ \partial count\text{-}space\ UNIV) = (\lambda i.\ \int y.\ f\text{-}i'\ i\ (y, x)\ \partial count\text{-}space\ UNIV)$
unfolding *sum-i′* **by**(*rule ext Bochner-Integration.integral-sum*[*symmetric*] *integrable-k′*)+
finally have $(\lambda i.\ ennreal\ (\int y.\ f\text{-}i'\ i\ (y, x)\ \partial count\text{-}space\ UNIV)) \longlongrightarrow ennreal\ (\int y.\ f'\ (y, x)\ \partial count\text{-}space\ UNIV)$ **by** *simp*
also have $(\lambda i.\ ennreal\ (\int y.\ f\text{-}i'\ i\ (y, x)\ \partial count\text{-}space\ UNIV)) = (\lambda i.\ d\text{-}IN\ (f\text{-}i\ i)\ x)$
unfolding *d-IN-def f-i′* **by**(*rule ext nn-integral-eq-integral*[*symmetric*] *integrable-f-i′*)+ *simp*
also have $ennreal\ (\int y.\ f'\ (y, x)\ \partial count\text{-}space\ UNIV) = d\text{-}IN\ f'\ x$
unfolding *d-IN-def* **by**(*rule nn-integral-eq-integral*[*symmetric*])(*simp-all add*: *f′-nonneg*, *simp add*: *suminf*[*symmetric*] *integrable-suminf integrable-k′ summable-nk summable′-k′*)
finally show $d\text{-}OUT\ f'\ x = d\text{-}IN\ f'\ x$ **using** $*$ **by**(*blast intro*: *LIMSEQ-unique*)
qed
moreover

{ **have** *incseq* (*λi. value-flow* Δ (*f-i i*))
by(*rule incseq-SucI*)(*simp add*: *value-aux value-f-i real field-simps α′-nonneg ennreal-leI del*: *f-i-simps*)
then have (*λi. value-flow* Δ (*f-i i*)) ⟶ (*SUP i. value-flow* Δ (*f-i i*)) **by**(*rule LIMSEQ-SUP*)
also have (*SUP i. value-flow* Δ (*f-i i*)) = *α*
proof −
have *α* − (*SUP i. value-flow* Δ (*f-i i*)) = (*INF i. α* − *value-flow* Δ (*f-i i*))
by(*simp add*: *ennreal-SUP-const-minus real*)
also have *α* − *value-flow* Δ (*f-i i*) = *α′* / *2* ^ *i* **for** *i*
by(*simp add*: *value-f-i value-aux real ennreal-minus-if field-simps mult-le-cancel-left1*)
hence (*INF i. α* − *value-flow* Δ (*f-i i*)) = (*INF i. ennreal* (*α′* / *2* ^ *i*))
by(*auto intro*: *INF-cong*)
also have ... = *0*
proof(*rule LIMSEQ-unique*)
show (*λi. α′* / *2* ^ *i*) ⟶ (*INF i. ennreal* (*α′* / *2* ^ *i*))
by(*rule LIMSEQ-INF*)(*simp add*: *field-simps real decseq-SucI*)
qed(*simp add*: *LIMSEQ-divide-realpow-zero real ennreal-0*[*symmetric*] *del*: *ennreal-0*)
finally show (*SUP i. value-flow* Δ (*f-i i*)) = *α*
apply (*intro antisym*)
apply (*auto simp*: *α-def intro*!: *SUP-mono fn.g*) []
apply (*rule ennreal-minus-eq-0*)
apply *assumption*
done
qed
also have (*λi. value-flow* Δ (*f-i i*)) ⟶ *value-flow* Δ *f′*
by(*simp add*: *value-flow*[*OF flow source-out*] *value-flow*[*OF fn.g source-out*] *f*)
ultimately have *value-flow* Δ *f′* = *α* **by**(*blast intro*: *LIMSEQ-unique*) }
note *value-f* = *this*
moreover {
fix *x*
have *d-IN f′ x* = ∫⁺ *y. liminf* (*λi. f-i i* (*y*, *x*)) ∂*count-space UNIV* **unfolding** *d-IN-def* **using** *f*
by(*simp add*: *tendsto-iff-Liminf-eq-Limsup*)
also have ... ≤ *liminf* (*λi. d-IN* (*f-i i*) *x*) **unfolding** *d-IN-def*
by(*rule nn-integral-liminf*)(*simp-all add*:)
also have ... ≤ *liminf* (*λi. α*) **using** *d-IN-i*[*of - x*] *fn.g*
by(*auto intro*!: *Liminf-mono SUP-upper2 eventually-sequentiallyI simp add*: *α-def*)
also have ... = *value-flow* Δ *f′* **using** *value-f* **by**(*simp add*: *Liminf-const*)
also note *calculation*
} **ultimately show** *?thesis* **by** *blast*
qed

theorem *ex-max-flow′′*: — eliminate assumption of no antiparallel edges using locale *wf-residual-network*
assumes *source-out*: ⋀*y. edge* Δ (*source* Δ) *y* ⟷ *y* = *x*

```
  and nontrivial: E ≠ {}
  and real: α = ennreal α′ and nn[simp]: 0 ≤ α′
  shows ∃f. flow Δ f ∧ value-flow Δ f = α ∧ (∀x. d-IN f x ≤ value-flow Δ f)
proof −
  interpret antiparallel-edges Δ ..
  interpret Δ′′: flow-attainability Δ′′
    by(rule Δ′′-flow-attainability flow-attainability.axioms(2))+unfold-locales
  have wf-Δ′′: Δ′′.wf-residual-network
    by(rule Δ′′-wf-residual-network; simp add: no-loop)

  have source-out′: edge Δ′′ (source Δ′′) y ⟷ y = Edge (source Δ) x for y
    by(auto simp add: source-out)
  have nontrivial′: V_Δ′′ − {source Δ′′, sink Δ′′} ≠ {} using nontrivial by(auto
simp add: V-Δ′′)

  have (SUP g ∈ {g. flow Δ′′ g}. value-flow Δ′′ g) = (SUP g ∈ {g. flow Δ g}.
value-flow Δ g) (is ?lhs = ?rhs)
  proof(intro antisym SUP-least; unfold mem-Collect-eq)
    fix g
    assume g: flow Δ′′ g
    hence value-flow Δ′′ g = value-flow Δ (collect g) by(simp add: value-collect)
    also { from g have flow Δ (collect g) by simp }
    then have … ≤ ?rhs by(blast intro: SUP-upper2)
    finally show value-flow Δ′′ g ≤ … .
  next
    fix g
    assume g: flow Δ g
    hence value-flow Δ g = value-flow Δ′′ (split g) by simp
    also { from g have flow Δ′′ (split g) by simp }
    then have … ≤ ?lhs by(blast intro: SUP-upper2)
    finally show value-flow Δ g ≤ ?lhs .
  qed
  with real have eq: (SUP g ∈ {g. flow Δ′′ g}. value-flow Δ′′ g) = ennreal α′
by(simp add: α-def)
  from Δ′′.ex-max-flow′[OF wf-Δ′′ source-out′ nontrivial′ eq]
  obtain f where f: flow Δ′′ f
    and value-flow Δ′′ f = α
    and IN: ⋀x. d-IN f x ≤ value-flow Δ′′ f unfolding eq real using nn by blast
  hence flow Δ (collect f) and value-flow Δ (collect f) = α by(simp-all add:
value-collect)
  moreover {
    fix x
    have d-IN (collect f) x = (∑⁺ y∈range (λy. Edge y x). f (y, Vertex x))
      by(simp add: nn-integral-count-space-reindex d-IN-def)
    also have … ≤ d-IN f (Vertex x) unfolding d-IN-def
      by (auto intro!: nn-integral-mono simp add: nn-integral-count-space-indicator
split: split-indicator)
    also have … ≤ value-flow Δ (collect f) using IN[of Vertex x] f by(simp add:
value-collect)
```

also note *calculation* **}**
ultimately show *?thesis* **by** *blast*
qed

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*
by(*cases x*) *auto*

private lemma *inj-on-Inner* [*simp*]: *inj-on Inner A*
by(*simp add*: *inj-on-def*)

private inductive *edge'* :: *'v node* ⇒ *'v node* ⇒ *bool*
where
SOURCE: *edge' SOURCE* (*Inner* (*source* Δ))
| *Inner*: *edge* Δ *x y* ⟹ *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* Δ *then* α *else 0*)
| *capacity'* (*Inner x*, *Inner y*) = *capacity* Δ (*x*, *y*)
| *capacity'* - = *0*

private lemma *capacity'-source-in* [*simp*]: *capacity'* (*y*, *Inner* (*source* Δ)) = (*if y* = *SOURCE then* α *else 0*)
by(*cases y*)(*simp-all add*: *capacity-outside source-in*)

private definition Δ' :: *'v node network*
where Δ' = ⦇*edge* = *edge'*, *capacity* = *capacity'*, *source* = *SOURCE*, *sink* = *Inner* (*sink* Δ)⦈

private lemma Δ'*-sel* [*simp*]:
edge Δ' = *edge'*
capacity Δ' = *capacity'*
source Δ' = *SOURCE*
sink Δ' = *Inner* (*sink* Δ)
by(*simp-all add*: Δ'*-def*)

private lemma E-Δ': $\mathbf{E}_{\Delta'}$ = {(*SOURCE*, *Inner* (*source* Δ))} ∪ (λ(*x*, *y*). (*Inner x*, *Inner y*)) ‘ **E**

by(*auto elim*: *edge*$'$*.cases*)

private lemma Δ'-*countable-network*:
  assumes $\alpha \neq \top$
  shows *countable-network* Δ'
proof
  show *countable* $\mathbf{E}_{\Delta'}$ **unfolding** $\mathbf{E}$-Δ' **by** *simp*
  show *source* $\Delta' \neq$ *sink* Δ' **by** *simp*
  show *capacity* Δ' $e = 0$ **if** $e \notin \mathbf{E}_{\Delta'}$ **for** e **using** *that* **unfolding** $\mathbf{E}$-Δ'
    by(*cases e rule*: *capacity*$'$*.cases*)(*auto intro*: *capacity-outside*)
  show *capacity* Δ' $e \neq \top$ **for** e **by**(*cases e rule*: *capacity*$'$*.cases*)(*simp-all add*: *assms*)
qed

private lemma Δ'-*flow-attainability*:
  assumes $\alpha \neq \top$
  shows *flow-attainability* Δ'
proof $-$
  interpret Δ': *countable-network* Δ' **using** *assms* **by**(*rule* Δ'-*countable-network*)
  show *?thesis*
  proof
    show *d-IN* (*capacity* Δ') $x \neq \top \vee$ *d-OUT* (*capacity* Δ') $x \neq \top$ **if** *sink*: $x \neq$ *sink* Δ' **for** x
    proof(*cases* x)
      case (*Inner* x')
      consider (*source*) x' = *source* Δ | (*IN*) $x' \neq$ *source* Δ *d-IN* (*capacity* Δ) $x' \neq \top$ | (*OUT*) *d-OUT* (*capacity* Δ) $x' \neq \top$
        using *finite-capacity*[*of* x'] *sink Inner* **by**(*auto*)
      thus *?thesis*
      proof(*cases*)
        case *source*
        with *Inner* **have** *d-IN* (*capacity* Δ') $x = (\sum^{+} y.\ \alpha *$ *indicator* {*SOURCE* :: $'$*v node*} *y*)
          unfolding *d-IN-def* **by**(*intro nn-integral-cong*)(*simp split*: *split-indicator*)
        also have $\ldots = \alpha$ **by**(*simp add*: *max-def*)
        finally show *?thesis* **using** *assms* **by** *simp*
      next
        case *IN*
        with *Inner* **have** *d-IN* (*capacity* Δ') $x = (\sum^{+} y \in$ *range Inner*. *capacity* Δ (*node.inner y*, x'))
        by(*auto simp add*: *d-IN-def nn-integral-count-space-indicator not-Inner-conv intro*!: *nn-integral-cong split*: *split-indicator*)
        also have $\ldots$ = *d-IN* (*capacity* Δ) x' **unfolding** *d-IN-def*
          by(*simp add*: *nn-integral-count-space-reindex*)
        finally show *?thesis* **using** *Inner sink IN* **by**(*simp*)
      next
        case *OUT*
        from *Inner* **have** *d-OUT* (*capacity* Δ') $x = (\sum^{+} y \in$ *range Inner*. *capacity* Δ (x', *node.inner y*))

by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator not-Inner-conv intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *d-OUT* (*capacity* Δ) *x′* **by**(*simp add*: *d-OUT-def nn-integral-count-space-reindex*)
finally show *?thesis* **using** *OUT* **by** *auto*
qed
qed(*simp add*: *d-IN-def*)
show ¬ *edge* Δ′ *x x* **for** *x* **by**(*cases x*)(*simp-all add*: *no-loop*)
show ¬ *edge* Δ′ *x* (*source* Δ′) **for** *x* **by** *simp*
qed
qed

private fun *lift* :: ′*v flow* ⇒ ′*v node flow*
where
lift f (*SOURCE*, *Inner y*) = (*if y* = *source* Δ *then value-flow* Δ *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 −
have *?lhs* = ($\sum^+$ *y*∈*range Inner*. *lift f* (*Inner x*, *y*))
by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator not-Inner-conv intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *?rhs* **by**(*simp add*: *nn-integral-count-space-reindex d-OUT-def*)
finally show *?thesis* **.**
qed

private lemma *d-OUT-lift-SOURCE* [*simp*]: *d-OUT* (*lift f*) *SOURCE* = *value-flow* Δ *f* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ($\sum^+$ *y*. *lift f* (*SOURCE*, *y*) ∗ *indicator* {*Inner* (*source* Δ)} *y*)
unfolding *d-OUT-def* **by**(*rule nn-integral-cong*)(*case-tac x*; *simp*)
also have ... = *?rhs* **by**(*simp add*: *nn-integral-count-space-indicator max-def*)
finally show *?thesis* **.**
qed

private lemma *d-IN-lift-Inner* [*simp*]:
assumes *x* ≠ *source* Δ
shows *d-IN* (*lift f*) (*Inner x*) = *d-IN f x* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = ($\sum^+$ *y*∈*range Inner*. *lift f* (*y*, *Inner x*)) **using** *assms*
by(*auto simp add*: *d-IN-def nn-integral-count-space-indicator not-Inner-conv intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *?rhs* **by**(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
finally show *?thesis* **.**
qed

private lemma *d-IN-lift-source* [*simp*]: *d-IN* (*lift f*) (*Inner* (*source* Δ)) = *value-flow* Δ *f* + *d-IN f* (*source* Δ) (**is** *?lhs* = *?rhs*)

proof −
 have *?lhs* = ($\sum^+$ *y*. *lift f* (*y*, *Inner* (*source* Δ)) ∗ *indicator* {*SOURCE*} *y*) + ($\sum^+$ *y*∈*range Inner*. *lift f* (*y*, *Inner* (*source* Δ)))
  (**is** - = *?SOURCE* + *?rest*)
  unfolding *d-IN-def*
  apply(*subst nn-integral-count-space-indicator*, *simp*)
  apply(*subst nn-integral-add*[*symmetric*])
  apply(*auto simp add*: *AE-count-space max-def not-Inner-conv split*: *split-indicator intro*!: *nn-integral-cong*)
  done
 also have *?rest* = *d-IN f* (*source* Δ) **by**(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
 also have *?SOURCE* = *value-flow* Δ *f*
  by(*simp add*: *max-def one-ennreal-def*[*symmetric*])
 finally show *?thesis* **.**
qed

private lemma *flow-lift* [*simp*]:
 assumes *flow* Δ *f*
 shows *flow* Δ′ (*lift f*)
proof
 show *lift f e* ≤ *capacity* Δ′ *e* **for** *e*
  by(*cases e rule*: *capacity′.cases*)(*auto intro*: *flowD-capacity*[*OF assms*] *simp add*: *α-def intro*: *SUP-upper2 assms*)

 fix *x*
 assume *x*: *x* ≠ *source* Δ′ *x* ≠ *sink* Δ′
 then obtain *x′* **where** *x′*: *x* = *Inner x′* **by**(*cases x*) *auto*
 then show *KIR* (*lift f*) *x* **using** *x*
  by(*cases x′* = *source* Δ)(*auto simp add*: *flowD-source-IN*[*OF assms*] *dest*: *flowD-KIR*[*OF assms*])
qed

private abbreviation (*input*) *unlift* :: ′*v node flow* ⇒ ′*v flow*
where *unlift f* ≡ (λ(*x*, *y*). *f* (*Inner x*, *Inner y*))

private lemma *flow-unlift* [*simp*]:
 assumes *f*: *flow* Δ′ *f*
 shows *flow* Δ (*unlift f*)
proof
 show *unlift f e* ≤ *capacity* Δ *e* **for** *e* **using** *flowD-capacity*[*OF f*, *of map-prod Inner Inner e*]
  by(*cases e*)(*simp*)
next
 fix *x*
 assume *x*: *x* ≠ *source* Δ *x* ≠ *sink* Δ
 have *d-OUT* (*unlift f*) *x* = ($\sum^+$ *y*∈*range Inner*. *f* (*Inner x*, *y*))
  by(*simp add*: *nn-integral-count-space-reindex d-OUT-def*)
 also have . . . = *d-OUT f* (*Inner x*) **using** *flowD-capacity*[*OF f*, *of* (*Inner x*,

SOURCE)]
by(*auto simp add*: *nn-integral-count-space-indicator d-OUT-def not-Inner-conv intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *d-IN f* (*Inner x*) **using** *x flowD-KIR*[*OF f*, *of Inner x*] **by**(*simp*)
also have ... = ($\sum^+$ *y*∈*range Inner*. *f* (*y*, *Inner x*))
using *x flowD-capacity*[*OF f*, *of* (*SOURCE*, *Inner x*)]
by(*auto simp add*: *nn-integral-count-space-indicator d-IN-def not-Inner-conv intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *d-IN* (*unlift f*) *x* **by**(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
finally show *KIR* (*unlift f*) *x* **.**
qed

private lemma *value-unlift*:
assumes *f*: *flow* Δ' *f*
shows *value-flow* Δ (*unlift f*) = *value-flow* Δ' *f*
proof −
have *value-flow* Δ (*unlift f*) = ($\sum^+$ *y*∈*range Inner*. *f* (*Inner* (*source* Δ), *y*))
by(*simp add*: *nn-integral-count-space-reindex d-OUT-def*)
also have ... = *d-OUT f* (*Inner* (*source* Δ)) **using** *flowD-capacity*[*OF f*, *of* (*Inner* (*source* Δ), *SOURCE*)]
by(*auto simp add*: *nn-integral-count-space-indicator d-OUT-def not-Inner-conv intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *d-IN f* (*Inner* (*source* Δ)) **using** *flowD-KIR*[*OF f*, *of Inner* (*source* Δ)] **by**(*simp*)
also have ... = ($\sum^+$ *y*. *f* (*y*, *Inner* (*source* Δ)) ∗ *indicator* {*SOURCE*} *y*)
unfolding *d-IN-def* **using** *flowD-capacity*[*OF f*, *of* (*x*, *Inner* (*source* Δ)) **for** *x*]
by(*intro nn-integral-cong*)(*auto intro*!: *antisym split*: *split-indicator if-split-asm elim*: *meta-allE*)
also have ... = *f* (*SOURCE*, *Inner* (*source* Δ)) **by** *simp*
also have ... = ($\sum^+$ *y*. *f* (*SOURCE*, *y*) ∗ *indicator* {*Inner* (*source* Δ)} *y*)
by(*simp add*: *one-ennreal-def*[*symmetric*])
also have ... = *value-flow* Δ' *f* **unfolding** *d-OUT-def*
unfolding *d-OUT-def* **using** *flowD-capacity*[*OF f*, *of* (*SOURCE*, *Inner x*) **for** *x*] *flowD-capacity*[*OF f*, *of* (*SOURCE*, *SOURCE*)]
apply(*intro nn-integral-cong*)
apply(*case-tac x*)
apply(*auto intro*!: *antisym split*: *split-indicator if-split-asm elim*: *meta-allE*)
done
finally show *?thesis* **.**
qed

theorem *ex-max-flow*:
$\exists f$. *flow* Δ *f* ∧ *value-flow* Δ *f* = α ∧ ($\forall x$. *d-IN f x* ≤ *value-flow* Δ *f*)
proof(*cases* α)
case (*real* α')
hence α: $\alpha \neq \top$ **by** *simp*
then interpret Δ': *flow-attainability* Δ' **by**(*rule* Δ'*-flow-attainability*)

have *source-out*: *edge* Δ' (*source* Δ') *y* $\longleftrightarrow$ *y* = *Inner* (*source* Δ) **for** *y* **by**(*auto*)
have *nontrivial*: $\mathbf{E}_{\Delta'} \neq \{\}$ **by**(*auto intro*: *edge'.intros*)

have *eq*: (*SUP g* $\in$ {*g*. *flow* Δ' *g*}. *value-flow* Δ' *g*) = (*SUP g* $\in$ {*g*. *flow* Δ *g*}. *value-flow* Δ *g*) (**is** *?lhs* = *?rhs*)
proof(*intro antisym SUP-least*; *unfold mem-Collect-eq*)
fix *g*
assume *g*: *flow* Δ' *g*
hence *value-flow* Δ' *g* = *value-flow* Δ (*unlift g*) **by**(*simp add*: *value-unlift*)
also { **from** *g* **have** *flow* Δ (*unlift g*) **by** *simp* }
then have ... $\leq$ *?rhs* **by**(*blast intro*: *SUP-upper2*)
finally show *value-flow* Δ' *g* $\leq$ ... **.**
next
fix *g*
assume *g*: *flow* Δ *g*
hence *value-flow* Δ *g* = *value-flow* Δ' (*lift g*) **by** *simp*
also { **from** *g* **have** *flow* Δ' (*lift g*) **by** *simp* }
then have ... $\leq$ *?lhs* **by**(*blast intro*: *SUP-upper2*)
finally show *value-flow* Δ *g* $\leq$ *?lhs* **.**
qed
also have ... = *ennreal* α' **using** *real* **by**(*simp add*: α*-def*)
finally obtain *f* **where** *f*: *flow* Δ' *f*
and *value-f*: *value-flow* Δ' *f* = ($\bigsqcup$*g*$\in${*g*. *flow* Δ' *g*}. *value-flow* Δ' *g*)
and *IN-f*: $\bigwedge$*x*. *d-IN f x* $\leq$ *value-flow* Δ' *f*
using ‹$0 \leq \alpha'$› **by**(*blast dest*: Δ'*.ex-max-flow''*[*OF source-out nontrivial*])
have *flow* Δ (*unlift f*) **using** *f* **by** *simp*
moreover have *value-flow* Δ (*unlift f*) = α **using** *f eq value-f* **by**(*simp add*: *value-unlift* α*-def*)
moreover {
fix *x*
have *d-IN* (*unlift f*) *x* = ($\sum^+$ *y*$\in$*range Inner*. *f* (*y*, *Inner x*))
by(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
also have ... $\leq$ *d-IN f* (*Inner x*) **unfolding** *d-IN-def*
by(*auto intro*!: *nn-integral-mono simp add*: *nn-integral-count-space-indicator split*: *split-indicator*)
also have ... $\leq$ *value-flow* Δ (*unlift f*) **using** *IN-f*[*of Inner x*] *f* **by**(*simp add*: *value-unlift*)
also note *calculation* }
ultimately show *?thesis* **by** *blast*
next
case *top*
show *?thesis*
proof(*cases* $\exists$*f*. *flow* Δ *f* $\wedge$ *value-flow* Δ *f* = $\top$)
case *True*
with *top* **show** *?thesis* **by** *auto*
next
case *False*
hence *real*: $\forall$*f*. α = $\top$ $\longrightarrow$ *flow* Δ *f* $\longrightarrow$ *value-flow* Δ *f* < α **using** *top* **by**

(*auto simp*: *less-top*)
{ **fix** *i*
have $2 * 2 \,\widehat{}\, i < \alpha$ **using** *top* **by** (*simp-all add*: *ennreal-mult-less-top power-less-top-ennreal*)
from *flow-by-value*[*OF this real*] **have** $\exists f.$ *flow* Δ $f \wedge$ *value-flow* Δ $f = 2 * 2 \,\widehat{}\, i$ **by** *blast* }
then obtain *f-i* **where** *f-i*: $\bigwedge i.$ *flow* Δ (*f-i i*)
and *value-i*: $\bigwedge i.$ *value-flow* Δ (*f-i i*) $= 2 * 2 \,\widehat{}\, i$ **by** *metis*
define *f* **where** $f\ e = (\sum^{+} i.$ *f-i i e* $/ (2 * 2 \,\widehat{}\, i))$ **for** *e*
have *flow* Δ *f*
proof
fix *e*
have $f\ e \leq (\sum^{+} i.$ (*SUP i. f-i i e*) $/ (2 * 2 \,\widehat{}\, i))$ **unfolding** *f-def*
by(*rule nn-integral-mono*)(*auto intro*!: *divide-right-mono-ennreal SUP-upper*)
also have $\ldots =$ (*SUP i. f-i i e*) $/ 2 * (\sum^{+} i.\ 1 / 2 \,\widehat{}\, i)$
apply(*subst nn-integral-cmult*[*symmetric*])
apply(*auto intro*!: *nn-integral-cong intro*: *SUP-upper2*
simp: *divide-ennreal-def ennreal-inverse-mult power-less-top-ennreal mult-ac*)
done
also have $(\sum^{+} i.\ 1 / 2 \,\widehat{}\, i) = (\sum i.$ *ennreal* $((1 / 2) \,\widehat{}\, i))$
by(*simp add*: *nn-integral-count-space-nat power-divide divide-ennreal*[*symmetric*] *ennreal-power*[*symmetric*])
also have $\ldots =$ *ennreal* $(\sum i.\ (1 / 2) \,\widehat{}\, i)$
by(*intro suminf-ennreal2 complete-algebra-summable-geometric*) *simp-all*
also have $\ldots = 2$ **by**(*subst suminf-geometric*; *simp*)
also have (*SUP i. f-i i e*) $/ 2 * 2 =$ (*SUP i. f-i i e*)
by (*simp add*: *ennreal-divide-times*)
also have $\ldots \leq$ *capacity* Δ *e* **by**(*rule SUP-least*)(*rule flowD-capacity*[*OF f-i*])
finally show $f\ e \leq$ *capacity* Δ *e* **.**

fix *x*
assume *x*: $x \neq$ *source* Δ $x \neq$ *sink* Δ
have *d-OUT f x* $= (\sum^{+} i \in UNIV.\ \sum^{+} y.$ *f-i i* $(x, y) / (2 * 2 \,\widehat{}\, i))$
unfolding *d-OUT-def f-def*
by(*subst nn-integral-snd-count-space*[**where** *f*=*case-prod* -, *simplified*])
(*simp add*: *nn-integral-fst-count-space*[**where** *f*=*case-prod* -, *simplified*])
also have $\ldots = (\sum^{+} i.$ *d-OUT* (*f-i i*) $x / (2 * 2 \,\widehat{}\, i))$ **unfolding** *d-OUT-def*
by(*simp add*: *nn-integral-divide*)
also have $\ldots = (\sum^{+} i.$ *d-IN* (*f-i i*) $x / (2 * 2 \,\widehat{}\, i))$ **by**(*simp add*: *flowD-KIR*[*OF f-i*, *OF x*])
also have $\ldots = (\sum^{+} i \in UNIV.\ \sum^{+} y.$ *f-i i* $(y, x) / (2 * 2 \,\widehat{}\, i))$
by(*simp add*: *nn-integral-divide d-IN-def*)
also have $\ldots =$ *d-IN f x* **unfolding** *d-IN-def f-def*
by(*subst nn-integral-snd-count-space*[**where** *f*=*case-prod* -, *simplified*])
(*simp add*: *nn-integral-fst-count-space*[**where** *f*=*case-prod* -, *simplified*])
finally show *KIR f x* **.**
qed
moreover {
have *value-flow* Δ $f = (\sum^{+} i.$ *value-flow* Δ (*f-i i*) $/ (2 * 2 \,\widehat{}\, i))$

unfolding *d-OUT-def f-def*
by(*subst nn-integral-snd-count-space*[**where** *f*=*case-prod -*, *simplified*])
(*simp add*: *nn-integral-fst-count-space*[**where** *f*=*case-prod -*, *simplified*]
nn-integral-divide[*symmetric*])
also have ... = ⊤
by(*simp add*: *value-i ennreal-mult-less-top power-less-top-ennreal*)
finally have *value-flow* Δ *f* = ⊤ **.**
}
ultimately show *?thesis* **using** *top* **by** *auto*
qed
qed

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* ∩ *SINK g*
by(*auto simp add*: *SINK.simps set-eq-iff d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0 add-eq-0-iff-both-eq-0*)

abbreviation *plus-web* :: (*'v*, *'more*) *web-scheme* ⇒ *'v current* ⇒ *'v current* ⇒ *'v current* (‹- ⌢ı -› [*66*, *66*] *65*)
where *plus-web* Γ *f g* ≡ *plus-current f* (*g* ↿ Γ / *f*)

lemma *d-OUT-plus-web*:
fixes Γ (**structure**)
shows *d-OUT* (*f* ⌢ *g*) *x* = *d-OUT f x* + *d-OUT* (*g* ↿ Γ / *f*) *x* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *d-OUT f x* + ($\sum^+$ *y*. (*if x* ∈ *RF*° (*TER f*) *then 0 else g* (*x*, *y*) ∗ *indicator* (− *RF* (*TER f*)) *y*))
unfolding *d-OUT-def* **by**(*subst nn-integral-add*[*symmetric*])(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *?rhs* **by**(*auto simp add*: *d-OUT-def intro*!: *arg-cong2*[**where**

f=(+)] *nn-integral-cong*)
finally show *?thesis* **.**
qed

lemma *d-IN-plus-web*:
fixes Γ (**structure**)
shows *d-IN* ($f \frown g$) *y* = *d-IN f y* + *d-IN* ($g \upharpoonright \Gamma$ / *f*) *y* (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *d-IN f y* + ($\sum^+$ *x*. (*if y* ∈ *RF* (*TER f*) *then 0 else g* (*x*, *y*) ∗ *indicator* (− RF° (*TER f*)) *x*))
unfolding *d-IN-def* **by**(*subst nn-integral-add*[*symmetric*])(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have . . . = *?rhs* **by**(*auto simp add*: *d-IN-def intro*!: *arg-cong2*[**where** *f*=(+)] *nn-integral-cong*)
finally show *?thesis* **.**
qed

lemma *plus-web-greater*: *f e* ≤ ($f \frown_\Gamma g$) *e*
by(*cases e*)(*auto split*: *split-indicator*)

lemma *current-plus-web*:
fixes Γ (**structure**)
shows ⟦ *current* Γ *f*; *wave* Γ *f*; *current* Γ *g* ⟧ ⟹ *current* Γ ($f \frown g$)
by(*blast intro*: *current-plus-current current-restrict-current*)

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* ∈ $\mathcal{E}$ (*TER f* ∪ *TER g*)
begin

qualified lemma *RF-f*: *x* ∉ RF° (*TER f*)
proof
assume ∗: *x* ∈ RF° (*TER f*)

from *x* **obtain** *p y* **where** *p*: *path* Γ *x p y* **and** *y*: *y* ∈ *B* Γ
and *bypass*: ⋀*z*. ⟦*x* ≠ *y*; *z* ∈ *set p*⟧ ⟹ *z* = *x* ∨ *z* ∉ *TER f* ∪ *TER g* **by**(*rule* $\mathcal{E}$-*E*) *blast*
from *rtrancl-path-distinct*[*OF p*] **obtain** *p*′
where *p*: *path* Γ *x p*′ *y* **and** *p*′: *set p*′ ⊆ *set p* **and** *distinct*: *distinct* (*x* # *p*′) **.**

from ∗ **have** *x*′: *x* ∈ *RF* (*TER f*) **and** $\mathcal{E}$: *x* ∉ $\mathcal{E}$ (*TER f*) **by**(*auto simp add*:

roofed-circ-def)
  hence $x \notin$ *TER f* **using** *not-essentialD*[*OF - p y*] p' *bypass* **by** *blast*
  with *roofedD*[*OF* x' *p y*] **obtain** *z* **where** *z*: $z \in$ *set* p' $z \in$ *TER f* **by** *auto*
  with *p* **have** $y \in$ *set* p' **by**(*auto dest*!: *rtrancl-path-last intro*: *last-in-set*)
  with *distinct* **have** $x \neq y$ **by** *auto*
  with *bypass z* p' *distinct* **show** *False* **by** *auto*
qed

private lemma *RF-g*: $x \notin RF^\circ$ (*TER g*)
proof
  assume $*$: $x \in RF^\circ$ (*TER g*)

  from *x* **obtain** *p y* **where** *p*: *path* Γ *x p y* **and** *y*: $y \in B\ \Gamma$
    and *bypass*: $\bigwedge z.$ $[\![x \neq y; z \in$ *set p*$]\!] \Longrightarrow z = x \vee z \notin$ *TER f* $\cup$ *TER g* **by**(*rule* $\mathcal{E}$-*E*) *blast*
  from *rtrancl-path-distinct*[*OF p*] **obtain** p'
    where *p*: *path* Γ *x* p' *y* **and** p': *set* $p' \subseteq$ *set p* **and** *distinct*: *distinct* $(x \# p')$ **.**

  from $*$ **have** x': $x \in RF$ (*TER g*) **and** $\mathcal{E}$: $x \notin \mathcal{E}$ (*TER g*) **by**(*auto simp add*: *roofed-circ-def*)
  hence $x \notin$ *TER g* **using** *not-essentialD*[*OF - p y*] p' *bypass* **by** *blast*
  with *roofedD*[*OF* x' *p y*] **obtain** *z* **where** *z*: $z \in$ *set* p' $z \in$ *TER g* **by** *auto*
  with *p* **have** $y \in$ *set* p' **by**(*auto dest*!: *rtrancl-path-last intro*: *last-in-set*)
  with *distinct* **have** $x \neq y$ **by** *auto*
  with *bypass z* p' *distinct* **show** *False* **by** *auto*
qed

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
  from *x* **obtain** *p y* **where** *p*: *path* Γ *x p y* **and** *y*: $y \in B\ \Gamma$
    and *bypass*: $\bigwedge z.$ $[\![x \neq y; z \in$ *set p*$]\!] \Longrightarrow z = x \vee z \notin$ *TER f* $\cup$ *TER g* **by**(*rule* $\mathcal{E}$-*E*) *blast*
  from *rtrancl-path-distinct*[*OF p*] **obtain** p'
    where *p*: *path* Γ *x* p' *y* **and** p': *set* $p' \subseteq$ *set p* **and** *distinct*: *distinct* $(x \# p')$ **.**

  from *RF-f* **have** $x \in$ *SINK f*
    by(*auto simp add*: *roofed-circ-def SINK.simps dest*: *waveD-OUT*[*OF w*])
  thus $x \in$ *SINK* $(f \frown g)$ **using** *SINK*
    by(*simp add*: *SINK.simps d-OUT-plus-web*)
  show $x \in$ *SAT* Γ $(f \frown g)$
  proof(*cases* $x \in$ *TER f*)
    case *True*
    hence $x \in$ *SAT* Γ *f* **by** *simp*
    moreover have $\ldots \subseteq$ *SAT* Γ $(f \frown g)$ **by**(*rule SAT-mono plus-web-greater*)+
    ultimately show *?thesis* **by** *blast*
  next
    case *False*

with *x* **have** $x \in$ *TER g* **by** *auto*
from *False RF-f* **have** $x \notin$ *RF* (*TER f*) **by**(*auto simp add*: *roofed-circ-def*)
moreover { **fix** *z*
assume *z*: $z \in RF^\circ$ (*TER f*)
have $(z, x) \notin$ **E**
proof
assume $(z, x) \in$ **E**
hence *path'*: *path* Γ *z* (x # p') *y* **using** *p* **by**(*simp add*: *rtrancl-path.step*)
from *z* **have** $z \in$ *RF* (*TER f*) **by**(*simp add*: *roofed-circ-def*)
from *roofedD*[*OF this path' y*] *False*
consider (*path*) *z'* **where** $z' \in$ *set p'* $z' \in$ *TER f* | (*TER*) $z \in$ *TER f* **by**
auto
then show *False*
proof *cases*
{ **case** (*path z'*)
with *p distinct* **have** $x \neq y$
by(*auto 4 3 intro*: *last-in-set elim*: *rtrancl-path.cases dest*: *rtrancl-path-last*[*symmetric*])
from *bypass*[*OF this*, *of z'*] *path False p'* **show** *False* **by** *auto* }
note *that* = *this*
case *TER*
with *z* **have** $\neg$ *essential* Γ (*B* Γ) (*TER f*) *z* **by**(*simp add*: *roofed-circ-def*)
from *not-essentialD*[*OF this path' y*] *False* **obtain** *z'* **where** $z' \in$ *set p'* *z'*
$\in$ *TER f* **by** *auto*
thus *False* **by**(*rule that*)
qed
qed }
ultimately have *d-IN ?g x* = *d-IN g x* **unfolding** *d-IN-def*
by(*intro nn-integral-cong*)(*clarsimp split*: *split-indicator simp add*: *currentD-outside*[*OF*
g])
hence *d-IN* ($f \frown g$) $x \geq$ *d-IN g x*
by(*simp add*: *d-IN-plus-web*)
with ‹$x \in$ *TER g*› **show** *?thesis* **by**(*auto elim*!: *SAT.cases intro*: *SAT.intros*)
qed
qed

qualified lemma *SINK-TER-in''*:
assumes $\bigwedge x.\ x \notin$ *RF* (*TER g*) $\Longrightarrow$ *d-OUT g x* = *0*
shows $x \in$ *SINK g*
using *RF-g* **by**(*auto simp add*: *roofed-circ-def SINK.simps assms*)

end

lemma *wave-plus*: *wave* (*quotient-web* Γ *f*) ($g \upharpoonright \Gamma$ / *f*) $\Longrightarrow$ *wave* Γ ($f \frown g$)
using *f w* **by**(*rule wave-plus-current*)(*rule current-restrict-current*[*OF w g*])

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

fix x
assume $*$: $x \in \mathcal{E}$ $(TER\ f \cup TER\ g)$
moreover have $x \in SINK$ $(g \upharpoonright \Gamma\ /\ f)$
by(*rule in-SINK-restrict-current*)(*rule MFMC-Unbounded.SINK-TER-in*$''$[*OF f w g* $*$ *assms*])
ultimately show $x \in TER$ $(f \frown g)$ **by**(*rule TER-plus-web-aux*)
qed

lemma *TER-plus-web*$'$: *wave* Γ $g \Longrightarrow \mathcal{E}$ $(TER\ f \cup TER\ g) \subseteq TER$ $(f \frown g)$
by(*rule TER-plus-web*$''$)(*rule waveD-OUT*)

lemma *wave-plus*$'$: *wave* Γ $g \Longrightarrow$ *wave* Γ $(f \frown g)$
by(*rule wave-plus*)(*rule wave-restrict-current*[*OF f w g*])

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 $(TER$ $(f \frown g)) = RF$ $(TER\ f \cup TER\ g)$
proof
have RF $(\mathcal{E}$ $(TER\ f \cup TER\ g)) \subseteq RF$ $(TER$ $(f \frown g))$
by(*rule roofed-mono*)(*rule TER-plus-web*$'$[*OF f w g w*$'$])
also have RF $(\mathcal{E}$ $(TER\ f \cup TER\ g)) = RF$ $(TER\ f \cup TER\ g)$ **by**(*rule RF-essential*)
finally show $\ldots \subseteq RF$ $(TER$ $(f \frown g))$ **.**
next
have fg: *current* Γ $(f \frown g)$ **using** f w g **by**(*rule current-plus-web*)
show RF $(TER$ $(f \frown g)) \subseteq RF$ $(TER\ f \cup TER\ g)$
proof(*intro subsetI roofedI*)
fix x p y
assume RF: $x \in RF$ $(TER$ $(f \frown g))$ **and** p: *path* Γ x p y **and** y: $y \in B$ Γ
from *roofedD*[*OF RF p y*] **obtain** z **where** z: $z \in$ *set* $(x\ \#\ p)$ **and** TER: $z \in TER$ $(f \frown g)$ **by** *auto*
from TER **have** $SINK$: $z \in SINK\ f$
by(*auto simp add*: *SINK.simps d-OUT-plus-web add-eq-0-iff-both-eq-0*)
from TER **have** $z \in SAT$ Γ $(f \frown g)$ **by** *simp*
hence SAT: $z \in SAT$ Γ $f \cup SAT$ Γ g
by(*cases* $z \in RF$ $(TER\ f)$)(*auto simp add*: *currentD-SAT*[*OF f*] *currentD-SAT*[*OF g*] *currentD-SAT*[*OF fg*] *d-IN-plus-web d-IN-restrict-current-outside restrict-current-IN-not-RF*[*OF g*] *wave-not-RF-IN-zero*[*OF f w*])

show $(\exists\, z {\in} set\ p.\ z \in TER\ f \cup TER\ g) \lor x \in TER\ f \cup TER\ g$
proof(*cases* $z \in RF$ $(TER\ g)$)
case *False*
hence $z \in SINK\ g$ **by**(*simp add*: *SINK.simps waveD-OUT*[*OF w*$'$])
with *SINK SAT* **have** $z \in TER\ f \cup TER\ g$ **by** *auto*

thus *?thesis* **using** *z* **by** *auto*
next
case *True*
from *split-list*[*OF z*] **obtain** *ys zs* **where** *split*: x # p = *ys* @ z # *zs* **by** *blast*
with *p* **have** *path* Γ *z zs y* **by**(*auto elim*: *rtrancl-path-appendE simp add*: *Cons-eq-append-conv*)
from *roofedD*[*OF True this y*] *split* **show** *?thesis* **by**(*auto simp add*: *Cons-eq-append-conv*)
qed
qed
qed

lemma *RF-TER-Sup*:
fixes Γ (**structure**)
assumes *f*: $\bigwedge f.\ f \in Y \Longrightarrow$ *current* Γ *f*
and *w*: $\bigwedge f.\ f \in Y \Longrightarrow$ *wave* Γ *f*
and *Y*: *Complete-Partial-Order.chain* (≤) *Y* $Y \neq \{\}$ *countable* (*support-flow* (*Sup Y*))
shows *RF* (*TER* (*Sup Y*)) = *RF* ($\bigcup f \in Y.$ *TER f*)
proof(*rule set-eqI iffI*)+
fix *x*
assume *x*: $x \in$ *RF* (*TER* (*Sup Y*))
have $x \in$ *RF* (*RF* ($\bigcup f \in Y.$ *TER f*))
proof
fix *p y*
assume *p*: *path* Γ *x p y* **and** *y*: $y \in B$ Γ
from *roofedD*[*OF x p y*] **obtain** *z* **where** *z*: $z \in$ *set* (x # p) **and** *TER*: $z \in$ *TER* (*Sup Y*) **by** *auto*
from *TER* **have** *SINK*: $z \in$ *SINK f* **if** $f \in Y$ **for** *f* **using** *that* **by**(*auto simp add*: *SINK-Sup*[*OF Y*])

from *Y*(*2*) **obtain** *f* **where** *y*: $f \in Y$ **by** *blast*

show ($\exists z \in$*set p*. $z \in$ *RF* ($\bigcup f \in Y.$ *TER f*)) ∨ $x \in$ *RF* ($\bigcup f \in Y.$ *TER f*)
proof(*cases* $\exists f \in Y.$ $z \in$ *RF* (*TER f*))
case *True*
then obtain *f* **where** *fY*: $f \in Y$ **and** *zf*: $z \in$ *RF* (*TER f*) **by** *blast*
from *zf* **have** $z \in$ *RF* ($\bigcup f \in Y.$ *TER f*) **by**(*rule in-roofed-mono*)(*auto intro*: *fY*)
with *z* **show** *?thesis* **by** *auto*
next
case *False*
hence ∗: *d-IN f z* = *0* **if** $f \in Y$ **for** *f* **using** *that* **by**(*auto intro*: *wave-not-RF-IN-zero*[*OF f w*])
hence *d-IN* (*Sup Y*) *z* = *0* **using** *Y*(*2*) **by**(*simp add*: *d-IN-Sup*[*OF Y*])
with *TER* **have** $z \in$ *SAT* Γ *f* **using** ∗[*OF y*]
by(*simp add*: *SAT.simps*)
with *SINK*[*OF y*] **have** $z \in$ *TER f* **by** *simp*
with *z y* **show** *?thesis* **by**(*auto intro*: *roofed-greaterI*)
qed

qed
then show $x \in RF$ ($\bigcup f{\in}Y.$ *TER f*) **unfolding** *roofed-idem* **.**
next
fix x
assume x: $x \in RF$ ($\bigcup f{\in}Y.$ *TER f*)
have $x \in RF$ (*RF* (*TER* ($\bigsqcup Y$)))
proof(*rule roofedI*)
fix p y
assume p: *path* Γ x p y **and** y: $y \in B\ \Gamma$
from *roofedD*[*OF x p y*] **obtain** z f **where** $*$: $z \in$ *set* (x # p)
and $**$: $f \in Y$ **and** *TER*: $z \in$ *TER f* **by** *auto*
have $z \in RF$ (*TER* (*Sup Y*))
proof(*rule ccontr*)
assume z: $z \notin RF$ (*TER* (*Sup Y*))
have *wave* Γ (*Sup Y*) **using** *Y*(*1*−*2*) *w* *Y*(*3*) **by**(*rule wave-lub*)
hence *d-OUT* (*Sup Y*) $z = 0$ **using** z **by**(*rule waveD-OUT*)
hence $z \in$ *SINK* (*Sup Y*) **by**(*simp add*: *SINK.simps*)
moreover have $z \in$ *SAT* Γ (*Sup Y*) **using** *TER SAT-Sup-upper*[*OF* $**$, *of* Γ] **by** *blast*
ultimately have $z \in$ *TER* (*Sup Y*) **by** *simp*
hence $z \in RF$ (*TER* (*Sup Y*)) **by**(*rule roofed-greaterI*)
with z **show** *False* **by** *contradiction*
qed
thus ($\exists z{\in}$*set p*. $z \in RF$ (*TER* (*Sup Y*))) $\vee$ $x \in RF$ (*TER* (*Sup Y*)) **using** $*$ **by** *auto*
qed
then show $x \in RF$ (*TER* ($\bigsqcup Y$)) **unfolding** *roofed-idem* **.**
qed

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-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*: $x \in A\ \Gamma \Longrightarrow u\ x = 0$
using *u-outside*[*of x*] *disjoint* **by** *auto*

private lemma *u-finite*: $u\ y \neq \top$
proof(*cases* $y \in B\ \Gamma$)
case *True*
have $u\ y \leq \varepsilon$ **unfolding** *ε-def* **by**(*rule nn-integral-ge-point*)(*simp add*: *True*)
also have $\ldots < \top$ **using** *finite-ε* **by** (*simp add*: *less-top*[*symmetric*])

finally show *?thesis* **by** *simp*
qed(*simp add*: *u-outside*)

lemma *hindered-reduce*: — Lemma 6.7
assumes *u*: $u \leq$ *weight* Γ
assumes *hindered-by*: *hindered-by* (Γ(|*weight* := *weight* Γ − *u*|)) ε (**is** *hindered-by* *?*Γ -)
shows *hindered* Γ
proof −
note [*simp*] = *u-finite*
let *?TER* = *TER*$_{?\Gamma}$
from *hindered-by* **obtain** *f*
where *hindrance-by*: *hindrance-by* *?*Γ *f* ε
and *f*: *current* *?*Γ *f*
and *w*: *wave* *?*Γ *f* **by** *cases*
from *hindrance-by* **obtain** *a* **where** *a*: $a \in A\ \Gamma\ a \notin \mathcal{E}_{?\Gamma}$ (*?TER f*)
and *a-le*: *d-OUT f a* < *weight* Γ *a*
and ε*-less*: *weight* Γ *a* − *d-OUT f a* > ε
and ε*-nonneg*: $\varepsilon \geq 0$ **by**(*auto simp add*: *u-A hindrance-by.simps*)

from *f* **have** *f'*: *current* Γ *f* **by**(*rule current-weight-mono*)(*auto intro*: *diff-le-self-ennreal*)

write *Some* (‹⟨-⟩›)

define *edge'*
where *edge' xo yo* =
(*case* (*xo*, *yo*) *of*
(*None*, *Some y*) ⇒ $y \in \mathbf{V} \wedge y \notin A\ \Gamma$
| (*Some x*, *Some y*) ⇒ *edge* Γ *x y* ∨ *edge* Γ *y x*
| - ⇒ *False*) **for** *xo yo*
define *cap*
where *cap e* =
(*case e of*
(*None*, *Some y*) ⇒ *if* $y \in \mathbf{V}$ *then u y else 0*
| (*Some x*, *Some y*) ⇒ *if edge* Γ *x y* ∧ *x* ≠ *a then f* (*x*, *y*) *else if edge* Γ *y x then max* (*weight* Γ *x*) (*weight* Γ *y*) + *1 else 0*
| - ⇒ *0*) **for** *e*
define Ψ **where** Ψ = (|*edge* = *edge'*, *capacity* = *cap*, *source* = *None*, *sink* = *Some a*|)

have *edge'-simps* [*simp*]:
edge' None ⟨*y*⟩ ⟷ $y \in \mathbf{V} \wedge y \notin A\ \Gamma$
edge' xo None ⟷ *False*
edge' ⟨*x*⟩ ⟨*y*⟩ ⟷ *edge* Γ *x y* ∨ *edge* Γ *y x*
for *xo x y* **by**(*simp-all add*: *edge'-def split*: *option.split*)

have *edge-None1E* [*elim*!]: *thesis* **if** *edge' None y* $\bigwedge z$. ⟦ *y* = ⟨*z*⟩; $z \in \mathbf{V}$; $z \notin A\ \Gamma$ ⟧ ⟹ *thesis* **for** *y thesis*
using *that* **by**(*simp add*: *edge'-def split*: *option.split-asm sum.split-asm*)

have *edge-Some1E* [*elim*!]: *thesis* **if** *edge′* $\langle x\rangle$ *y* $\bigwedge z.$ $[\![$ $y = \langle z\rangle$; *edge* Γ *x z* $\vee$ *edge* Γ *z x* $]\!]$ $\Longrightarrow$ *thesis* **for** *x y thesis*
using *that* **by**(*simp add*: *edge′-def split*: *option.split-asm sum.split-asm*)
have *edge-Some2E* [*elim*!]: *thesis* **if** *edge′ x* $\langle y\rangle$ $[\![$ $x = None$; $y \in \mathbf{V}$; $y \notin A\ \Gamma$ $]\!]$ $\Longrightarrow$ *thesis* $\bigwedge z.$ $[\![$ $x = \langle z\rangle$; *edge* Γ *z y* $\vee$ *edge* Γ *y z* $]\!]$ $\Longrightarrow$ *thesis* **for** *x y thesis*
using *that* **by**(*simp add*: *edge′-def split*: *option.split-asm sum.split-asm*)

have *cap-simps* [*simp*]:
cap (*None*, $\langle y\rangle$) = (*if* $y \in \mathbf{V}$ *then u y else 0*)
cap (*xo*, *None*) = *0*
cap ($\langle x\rangle$, $\langle y\rangle$) =
(*if edge* Γ *x y* $\wedge$ $x \neq a$ *then f* (*x*, *y*) *else if edge* Γ *y x then max* (*weight* Γ *x*) (*weight* Γ *y*) + *1 else 0*)
for *xo x y* **by**(*simp-all add*: *cap-def split*: *option.split*)

have Ψ*-sel* [*simp*]:
edge Ψ = *edge′*
capacity Ψ = *cap*
source Ψ = *None*
sink Ψ = $\langle a\rangle$
by(*simp-all add*: Ψ*-def*)

have *cap-outside1*: $\neg$ *vertex* Γ *x* $\Longrightarrow$ *cap* ($\langle x\rangle$, *y*) = *0* **for** *x y*
by(*cases y*)(*auto simp add*: *vertex-def*)
have *capacity-A-weight*: *d-OUT cap* $\langle x\rangle$ $\leq$ *weight* Γ *x* **if** $x \in A\ \Gamma$ **for** *x*
proof −
have *d-OUT cap* $\langle x\rangle \leq (\sum^{+} y{\in}range\ Some.\ f\ (x,\ the\ y))$
using *that disjoint a*(*1*) **unfolding** *d-OUT-def*
by(*auto 4 4 intro*!: *nn-integral-mono simp add*: *nn-integral-count-space-indicator notin-range-Some currentD-outside*[*OF f*] *split*: *split-indicator dest*: *edge-antiparallel bipartite-E*)
also have ... = *d-OUT f x* **by**(*simp add*: *d-OUT-def nn-integral-count-space-reindex*)
also have ... $\leq$ *weight* Γ *x* **using** *f′* **by**(*rule currentD-weight-OUT*)
finally show *?thesis* **.**
qed
have *flow-attainability*: *flow-attainability* Ψ
proof
have $\mathbf{E}_\Psi$ = (λ(*x*, *y*). ($\langle x\rangle$, $\langle y\rangle$)) ‘ $\mathbf{E}$ $\cup$ (λ(*x*, *y*). ($\langle y\rangle$, $\langle x\rangle$)) ‘ $\mathbf{E}$ $\cup$ ($\lambda x.$ (*None*, $\langle x\rangle$)) ‘ ($\mathbf{V}$ $\cap$ − *A* Γ)
by(*auto simp add*: *edge′-def split*: *option.split-asm*)
thus *countable* $\mathbf{E}_\Psi$ **by** *simp*
next
fix *v*
assume $v \neq$ *sink* Ψ
consider (*sink*) *v* = *None* | (*A*) *x* **where** *v* = $\langle x\rangle$ $x \in A\ \Gamma$
| (*B*) *y* **where** *v* = $\langle y\rangle$ $y \notin A\ \Gamma$ $y \in \mathbf{V}$ | (*outside*) *x* **where** *v* = $\langle x\rangle$ $x \notin \mathbf{V}$
by(*cases v*) *auto*
then show *d-IN* (*capacity* Ψ) *v* $\neq \top$ $\vee$ *d-OUT* (*capacity* Ψ) *v* $\neq \top$
proof *cases*

case *sink* **thus** *?thesis* **by**(*simp add*: *d-IN-def*)
next
case (*A x*)
thus *?thesis* **using** *capacity-A-weight*[*of x*] **by** (*auto simp*: *top-unique*)
next
case (*B y*)
have *d-IN* (*capacity* Ψ) *v* ≤ ($\sum^+$ *x*. *f* (*the x*, *y*) ∗ *indicator* (*range Some*) *x* + *u y* ∗ *indicator* {*None*} *x*)
using *B disjoint bipartite-V a*(*1*) **unfolding** *d-IN-def*
by(*auto 4 4 intro*!: *nn-integral-mono simp add*: *nn-integral-count-space-indicator notin-range-Some currentD-outside*[*OF f*] *split*: *split-indicator dest*: *edge-antiparallel bipartite-E*)
also have . . . = ($\sum^+$ *x*∈*range Some*. *f* (*the x*, *y*)) + *u y*
by(*subst nn-integral-add*)(*simp-all add*: *nn-integral-count-space-indicator*)
also have . . . = *d-IN f y* + *u y* **by**(*simp add*: *d-IN-def nn-integral-count-space-reindex*)
also have *d-IN f y* ≤ *weight* Γ *y* **using** *f′* **by**(*rule currentD-weight-IN*)
finally show *?thesis* **by**(*auto simp add*: *add-right-mono top-unique split*: *if-split-asm*)
next
case *outside*
hence *d-OUT* (*capacity* Ψ) *v* = *0*
by(*auto simp add*: *d-OUT-def nn-integral-0-iff-AE AE-count-space cap-def vertex-def split*: *option.split*)
thus *?thesis* **by** *simp*
qed
next
show *capacity* Ψ *e* ≠ ⊤ **for** *e* **using** *weight-finite*
by(*auto simp add*: *cap-def max-def vertex-def currentD-finite*[*OF f′*] *split*: *option.split prod.split simp del*: *weight-finite*)
show *capacity* Ψ *e* = *0* **if** *e* ∉ $\mathbf{E}_{\Psi}$ **for** *e*
using *that bipartite-V disjoint*
by(*auto simp add*: *cap-def max-def intro*: *u-outside split*: *option.split prod.split*)
show ¬ *edge* Ψ *x* (*source* Ψ) **for** *x* **by** *simp*
show ¬ *edge* Ψ *x x* **for** *x* **by**(*cases x*)(*simp-all add*: *no-loop*)
show *source* Ψ ≠ *sink* Ψ **by** *simp*
qed
then interpret Ψ: *flow-attainability* Ψ **.**

define α **where** α = ($\bigsqcup$*g*∈{*g*. *flow* Ψ *g*}. *value-flow* Ψ *g*)
have *α-le*: α ≤ ε
proof −
have α ≤ *d-OUT cap None* **unfolding** *α-def* **by**(*rule SUP-least*)(*auto intro*!: *d-OUT-mono dest*: *flowD-capacity*)
also have . . . ≤ $\int^+$ *y*. *cap* (*None*, *y*) ∂*count-space* (*range Some*) **unfolding** *d-OUT-def*
by(*auto simp add*: *nn-integral-count-space-indicator notin-range-Some intro*!: *nn-integral-mono split*: *split-indicator*)
also have . . . ≤ ε **unfolding** *ε-def*
by (*subst* (*2*) *nn-integral-count-space-indicator*, *auto simp add*: *nn-integral-count-space-reindex*

u-outside intro!: *nn-integral-mono split*: *split-indicator*)
  finally show *?thesis* **by** *simp*
qed
then have *finite-flow*: $\alpha \neq \top$ **using** *finite-ε* **by** (*auto simp*: *top-unique*)

from Ψ.*ex-max-flow*
obtain *j* **where** *j*: *flow* Ψ *j*
  and *value-j*: *value-flow* Ψ *j* = α
  and *IN-j*: $\bigwedge x$. *d-IN j x* ≤ α
  unfolding *α-def* **by** *auto*

have *j-le-f*: *j* (*Some x*, *Some y*) ≤ *f* (*x*, *y*) **if** *edge* Γ *x y* **for** *x y*
  using *that flowD-capacity*[*OF j*, *of* (*Some x*, *Some y*)] *a*(*1*) *disjoint*
  by(*auto split*: *if-split-asm dest*: *bipartite-E intro*: *order-trans*)

have *IN-j-finite* [*simp*]: *d-IN j x* ≠ ⊤ **for** *x* **using** *finite-flow* **by**(*rule neq-top-trans*)(*simp add*: *IN-j*)

have *j-finite*[*simp*]: *j* (*x*, *y*) < ⊤ **for** *x y*
  by (*rule le-less-trans*[*OF d-IN-ge-point*]) (*simp add*: *IN-j-finite*[*of y*] *less-top*[*symmetric*])

have *OUT-j-finite*: *d-OUT j x* ≠ ⊤ **for** *x*
proof(*cases x* = *source* Ψ ∨ *x* = *sink* Ψ)
  case *True* **thus** *?thesis*
  proof *cases*
    case *left* **thus** *?thesis* **using** *finite-flow value-j* **by** *simp*
  next
    case *right*
    have *d-OUT* (*capacity* Ψ) ⟨*a*⟩ ≠ ⊤ **using** *capacity-A-weight*[*of a*] *a*(*1*) **by**(*auto simp*: *top-unique*)
    thus *?thesis* **unfolding** *right*[*simplified*]
      by(*rule neq-top-trans*)(*rule d-OUT-mono flowD-capacity*[*OF j*])+
  qed
next
  case *False* **then show** *?thesis* **by**(*simp add*: *flowD-KIR*[*OF j*])
qed

have *IN-j-le-weight*: *d-IN j* ⟨*x*⟩ ≤ *weight* Γ *x* **for** *x*
proof(*cases x* ∈ *A* Γ)
  case *xA*: *True*
  show *?thesis*
  proof(*cases x* = *a*)
    case *True*
    have *d-IN j* ⟨*x*⟩ ≤ α **by**(*rule IN-j*)
    also have ... ≤ ε **by**(*rule α-le*)
    also have ε < *weight* Γ *a* **using** *ε-less diff-le-self-ennreal less-le-trans* **by** *blast*
    finally show *?thesis* **using** *True* **by**(*auto intro*: *order.strict-implies-order*)
  next
    case *False*

have *d-IN j* $\langle x\rangle$ = *d-OUT j* $\langle x\rangle$ **using** *flowD-KIR*[*OF j*, *of Some x*] *False* **by** *simp*
also have ... $\leq$ *d-OUT cap* $\langle x\rangle$ **using** *flowD-capacity*[*OF j*] **by**(*auto intro*: *d-OUT-mono*)
also have ... $\leq$ *weight* Γ *x* **using** *xA* **by**(*rule capacity-A-weight*)
finally show *?thesis* **.**
qed
next
case *xA*: *False*
show *?thesis*
proof(*cases* $x \in B\ \Gamma$)
case *True*
have *d-IN j* $\langle x\rangle \leq$ *d-IN cap* $\langle x\rangle$ **using** *flowD-capacity*[*OF j*] **by**(*auto intro*: *d-IN-mono*)
also have ... $\leq$ ($\sum^+$ *z*. *f* (*the z*, *x*) $*$ *indicator* (*range Some*) *z*) + ($\sum^+$ *z* :: *'v option*. *u x* $*$ *indicator* {*None*} *z*)
using *True disjoint*
by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *vertex-def currentD-outside*[*OF f*] *d-IN-def B-out intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... = *d-IN f x* + *u x*
by(*simp add*: *nn-integral-count-space-indicator*[*symmetric*] *nn-integral-count-space-reindex d-IN-def*)
also have ... $\leq$ *weight* Γ *x* **using** *currentD-weight-IN*[*OF f*, *of x*] *u-finite*[*of x*]
using ε*-less u* **by** (*auto simp add*: *ennreal-le-minus-iff le-fun-def*)
finally show *?thesis* **.**
next
case *False*
with *xA* **have** $x \notin$ **V** **using** *bipartite-V* **by** *blast*
then have *d-IN j* $\langle x\rangle$ = *0* **using** *False*
by(*auto simp add*: *d-IN-def nn-integral-0-iff emeasure-count-space-eq-0 vertex-def edge'-def split*: *option.split-asm intro*!: Ψ.*flowD-outside*[*OF j*])
then show *?thesis*
by *simp*
qed
qed

let *?j* = *j* $\circ$ *map-prod Some Some* $\circ$ *prod.swap*

have *finite-j-OUT*: ($\sum^+$ *y*$\in$**OUT** *x*. *j* ($\langle x\rangle$, $\langle y\rangle$)) $\neq \top$ (**is** *?j-OUT* $\neq$ -) **if** $x \in A\ \Gamma$ **for** *x*
using *currentD-finite-OUT*[*OF f'*, *of x*]
by(*rule neq-top-trans*)(*auto intro*!: *nn-integral-mono j-le-f simp add*: *d-OUT-def nn-integral-count-space-indicator outgoing-def split*: *split-indicator*)
have *j-OUT-eq*: *?j-OUT x* = *d-OUT j* $\langle x\rangle$ **if** $x \in A\ \Gamma$ **for** *x*
proof −
have *?j-OUT x* = ($\sum^+$ *y*$\in$*range Some*. *j* (*Some x*, *y*)) **using** *that disjoint*
by(*simp add*: *nn-integral-count-space-reindex*)(*auto 4 4 simp add*: *nn-integral-count-space-indicator outgoing-def intro*!: *nn-integral-cong* Ψ.*flowD-outside*[*OF j*] *dest*: *bipartite-E split*:

split-indicator)
  also have ... $=$ *d-OUT j* $\langle x\rangle$
  by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator notin-range-Some intro*!: *nn-integral-cong* Ψ*.flowD-outside*[*OF j*] *split*: *split-indicator*)
  finally show *?thesis* **.**
 qed

 define *g* **where** $g = f \oplus$ *?j*
 have *g-simps*: $g\ (x,\ y) = (f \oplus$ *?j*$)\ (x,\ y)$ **for** *x y* **by**(*simp add*: *g-def*)

 have *OUT-g-A*: *d-OUT g x* $=$ *d-OUT f x* $+$ *d-IN j* $\langle x\rangle$ $-$ *d-OUT j* $\langle x\rangle$ **if** $x \in A$ Γ **for** *x*
 proof $-$
  have *d-OUT g x* $= (\sum^+ y{\in}\mathbf{OUT}\ x.\ f\ (x,\ y) + j\ (\langle y\rangle,\ \langle x\rangle) - j\ (\langle x\rangle,\ \langle y\rangle))$
   by(*auto simp add*: *d-OUT-def g-simps currentD-outside*[*OF f'*] *outgoing-def nn-integral-count-space-indicator intro*!: *nn-integral-cong*)
  also have ... $= (\sum^+ y{\in}\mathbf{OUT}\ x.\ f\ (x,\ y) + j\ (\langle y\rangle,\ \langle x\rangle)) - (\sum^+ y{\in}\mathbf{OUT}\ x.\ j\ (\langle x\rangle,\ \langle y\rangle))$
   (**is** - = - $-$ *?j-OUT*) **using** *finite-j-OUT*[*OF that*]
   by(*subst nn-integral-diff*)(*auto simp add*: *AE-count-space outgoing-def intro*!: *order-trans*[*OF j-le-f*])
  also have ... $= (\sum^+ y{\in}\mathbf{OUT}\ x.\ f\ (x,\ y)) + (\sum^+ y{\in}\mathbf{OUT}\ x.\ j$ (*Some y*, *Some x*)) $-$ *?j-OUT*
   (**is** - = *?f* + *?j-IN* $-$ -) **by**(*subst nn-integral-add*) *simp-all*
  also have *?f* $=$ *d-OUT f x* **by**(*subst d-OUT-alt-def*[**where** $G{=}\Gamma$])(*simp-all add*: *currentD-outside*[*OF f*])
  also have *?j-OUT* $=$ *d-OUT j* $\langle x\rangle$ **using** *that* **by**(*rule j-OUT-eq*)
  also have *?j-IN* $= (\sum^+ y{\in}$*range Some*. $j\ (y,\ \langle x\rangle))$ **using** *that disjoint*
  by(*simp add*: *nn-integral-count-space-reindex*)(*auto 4 4 simp add*: *nn-integral-count-space-indicator outgoing-def intro*!: *nn-integral-cong* Ψ*.flowD-outside*[*OF j*] *split*: *split-indicator dest*: *bipartite-E*)
  also have ... $=$ *d-IN j* (*Some x*) **using** *that disjoint*
  by(*auto 4 3 simp add*: *d-IN-def nn-integral-count-space-indicator notin-range-Some intro*!: *nn-integral-cong* Ψ*.flowD-outside*[*OF j*] *split*: *split-indicator*)
  finally show *?thesis* **by** *simp*
 qed

 have *OUT-g-B*: *d-OUT g x* $=$ *0* **if** $x \notin A\ \Gamma$ **for** *x*
  using *disjoint that*
  by(*auto simp add*: *d-OUT-def nn-integral-0-iff-AE AE-count-space g-simps dest*: *bipartite-E*)

 have *OUT-g-a*: *d-OUT g a* $<$ *weight* Γ *a* **using** *a*(*1*)
 proof $-$
  have *d-OUT g a* $=$ *d-OUT f a* $+$ *d-IN j* $\langle a\rangle$ $-$ *d-OUT j* $\langle a\rangle$ **using** *a*(*1*) **by**(*rule OUT-g-A*)
  also have ... $\leq$ *d-OUT f a* $+$ *d-IN j* $\langle a\rangle$
   by(*rule diff-le-self-ennreal*)
  also have ... $<$ *weight* Γ *a* $+$ *d-IN j* $\langle a\rangle$ $- \varepsilon$

using *finite-ε ε-less currentD-finite-OUT*[*OF f'*]
by (*simp add*: *less-diff-eq-ennreal less-top ac-simps*)
also have ... $\leq$ *weight* Γ *a*
using *IN-j*[*THEN order-trans*, *OF α-le*] **by** (*simp add*: *ennreal-minus-le-iff*)
finally show *?thesis* **.**
qed

have *OUT-jj*: *d-OUT ?j x* = *d-IN j* ⟨*x*⟩ − *j* (*None*, ⟨*x*⟩) **for** *x*
proof −
have *d-OUT ?j x* = ($\sum^+$ *y*∈*range Some*. *j* (*y*, ⟨*x*⟩)) **by**(*simp add*: *d-OUT-def nn-integral-count-space-reindex*)
also have ... = *d-IN j* ⟨*x*⟩ − ($\sum^+$ *y*. *j* (*y*, ⟨*x*⟩) ∗ *indicator* {*None*} *y*) **unfolding** *d-IN-def*
by(*subst nn-integral-diff*[*symmetric*])(*auto simp add*: *max-def* Ψ.*flowD-finite*[*OF j*] *AE-count-space nn-integral-count-space-indicator split*: *split-indicator intro*!: *nn-integral-cong*)
also have ... = *d-IN j* ⟨*x*⟩ − *j* (*None*, ⟨*x*⟩) **by**(*simp add*: *max-def*)
finally show *?thesis* **.**
qed

have *OUT-jj-finite* [*simp*]: *d-OUT ?j x* ≠ ⊤ **for** *x*
by(*simp add*: *OUT-jj*)

have *IN-g*: *d-IN g x* = *d-IN f x* + *j* (*None*, ⟨*x*⟩) **for** *x*
proof(*cases x* ∈ *B* Γ)
case *True*
have *finite*: ($\sum^+$ *y*∈**IN** *x*. *j* (*Some y*, *Some x*)) ≠ ⊤ **using** *currentD-finite-IN*[*OF f'*, *of x*]
by(*rule neq-top-trans*)(*auto intro*!: *nn-integral-mono j-le-f simp add*: *d-IN-def nn-integral-count-space-indicator incoming-def split*: *split-indicator*)

have *d-IN g x* = *d-IN* (*f* ⊕ *?j*) *x* **by**(*simp add*: *g-def*)
also have ... = ($\sum^+$ *y*∈**IN** *x*. *f* (*y*, *x*) + *j* (*Some x*, *Some y*) − *j* (*Some y*, *Some x*))
by(*auto simp add*: *d-IN-def currentD-outside*[*OF f'*] *incoming-def nn-integral-count-space-indicator intro*!: *nn-integral-cong*)
also have ... = ($\sum^+$ *y*∈**IN** *x*. *f* (*y*, *x*) + *j* (*Some x*, *Some y*)) − ($\sum^+$ *y*∈**IN** *x*. *j* (*Some y*, *Some x*))
(**is** - = - − *?j-IN*) **using** *finite*
by(*subst nn-integral-diff*)(*auto simp add*: *AE-count-space incoming-def intro*!: *order-trans*[*OF j-le-f*])
also have ... = ($\sum^+$ *y*∈**IN** *x*. *f* (*y*, *x*)) + ($\sum^+$ *y*∈**IN** *x*. *j* (*Some x*, *Some y*)) − *?j-IN*
(**is** - = *?f* + *?j-OUT* − -) **by**(*subst nn-integral-add*) *simp-all*
also have *?f* = *d-IN f x* **by**(*subst d-IN-alt-def*[**where** *G*=Γ])(*simp-all add*: *currentD-outside*[*OF f*])
also have *?j-OUT* = ($\sum^+$ *y*∈*range Some*. *j* (*Some x*, *y*)) **using** *True disjoint*
by(*simp add*: *nn-integral-count-space-reindex*)(*auto 4 4 simp add*: *nn-integral-count-space-indicator incoming-def intro*!: *nn-integral-cong* Ψ.*flowD-outside*[*OF j*] *split*: *split-indicator dest*: *bipartite-E*)

```
    also have ... = d-OUT j (Some x) using disjoint
    by(auto 4 3 simp add: d-OUT-def nn-integral-count-space-indicator notin-range-Some
intro!: nn-integral-cong Ψ.flowD-outside[OF j] split: split-indicator)
    also have ... = d-IN j (Some x) using flowD-KIR[OF j, of Some x] True a
disjoint by auto
    also have ?j-IN = (∑+ y∈range Some. j (y, Some x)) using True disjoint
    by(simp add: nn-integral-count-space-reindex)(auto 4 4 simp add: nn-integral-count-space-indicator
incoming-def intro!: nn-integral-cong Ψ.flowD-outside[OF j] dest: bipartite-E split:
split-indicator)
    also have ... = d-IN j (Some x) − (∑+ y :: 'v option. j (None, Some x) ∗
indicator {None} y)
      unfolding d-IN-def using flowD-capacity[OF j, of (None, Some x)]
      by(subst nn-integral-diff[symmetric])
        (auto simp add: nn-integral-count-space-indicator AE-count-space top-unique
image-iff
            intro!: nn-integral-cong ennreal-diff-self split: split-indicator if-split-asm)
    also have d-IN f x + d-IN j (Some x) − ... = d-IN f x + j (None, Some x)
      using ennreal-add-diff-cancel-right[OF IN-j-finite[of Some x], of d-IN f x + j
(None, Some x)]
      apply(subst diff-diff-ennreal')
    apply(auto simp add: d-IN-def intro!: nn-integral-ge-point ennreal-diff-le-mono-left)
      apply(simp add: ac-simps)
      done
    finally show ?thesis .
  next
    case False
    hence d-IN g x = 0 d-IN f x = 0 j (None, ⟨x⟩) = 0
      using disjoint currentD-IN[OF f', of x] bipartite-V currentD-outside-IN[OF
f'] u-outside[OF False] flowD-capacity[OF j, of (None, ⟨x⟩)]
    by(cases vertex Γ x; auto simp add: d-IN-def nn-integral-0-iff-AE AE-count-space
g-simps dest: bipartite-E split: if-split-asm)+
    thus ?thesis by simp
  qed

  have g: current Γ g
  proof
    show d-OUT g x ≤ weight Γ x for x
    proof(cases x ∈ A Γ)
      case False
      thus ?thesis by(simp add: OUT-g-B)
    next
      case True
      with OUT-g-a show ?thesis
      by(cases x = a)(simp-all add: OUT-g-A flowD-KIR[OF j] currentD-weight-OUT[OF
f'])
    qed

    show d-IN g x ≤ weight Γ x for x
    proof(cases x ∈ B Γ)
```

case *False*
hence *d-IN g x* = *0* **using** *disjoint*
by(*auto simp add*: *d-IN-def nn-integral-0-iff-AE AE-count-space g-simps dest*: *bipartite-E*)
thus *?thesis* **by** *simp*
next
case *True*
have *d-IN g x* ≤ (*weight* Γ *x* − *u x*) + *u x* **unfolding** *IN-g*
using *currentD-weight-IN*[*OF f*, *of x*] *flowD-capacity*[*OF j*, *of* (*None*, *Some x*)] *True bipartite-V*
by(*intro add-mono*)(*simp-all split*: *if-split-asm*)
also have . . . = *weight* Γ *x*
using *u* **by** (*intro diff-add-cancel-ennreal*) (*simp add*: *le-fun-def*)
finally show *?thesis* **.**
qed
show *g e* = *0* **if** *e* ∉ **E** **for** *e* **using** *that*
by(*cases e*)(*auto simp add*: *g-simps*)
qed

define *cap′* **where** *cap′* = (λ(*x*, *y*). *if edge* Γ *x y then g* (*x*, *y*) *else if edge* Γ *y x then 1 else 0*)

have *cap′-simps* [*simp*]: *cap′* (*x*, *y*) = (*if edge* Γ *x y then g* (*x*, *y*) *else if edge* Γ *y x then 1 else 0*)
for *x y* **by**(*simp add*: *cap′-def*)

define *G* **where** *G* = (|*edge* = λ*x y*. *cap′* (*x*, *y*) > *0*|)
have *G-sel* [*simp*]: *edge G x y* ⟷ *cap′* (*x*, *y*) > *0* **for** *x y* **by**(*simp add*: *G-def*)
define *reachable* **where** *reachable x* = (*edge G*)** *x a* **for** *x*
have *reachable-alt-def*: *reachable* ≡ λ*x*. ∃ *p*. *path G x p a*
by(*simp add*: *reachable-def* [*abs-def*] *rtranclp-eq-rtrancl-path*)

have [*simp*]: *reachable a* **by**(*auto simp add*: *reachable-def*)

have *AB-edge*: *edge G x y* **if** *edge* Γ *y x* **for** *x y*
using *that*
by(*auto dest*: *edge-antiparallel simp add*: *min-def le-neq-trans add-eq-0-iff-both-eq-0*)
have *reachable-AB*: *reachable y* **if** *reachable x* (*x*, *y*) ∈ **E** **for** *x y*
using *that* **by**(*auto simp add*: *reachable-def simp del*: *G-sel dest*!: *AB-edge intro*: *rtrancl-path.step*)
have *reachable-BA*: *g* (*x*, *y*) = *0* **if** *reachable y* (*x*, *y*) ∈ **E** ¬ *reachable x* **for** *x y*
proof(*rule ccontr*)
assume *g* (*x*, *y*) ≠ *0*
then have *g* (*x*, *y*) > *0* **by** (*simp add*: *zero-less-iff-neq-zero*)
hence *edge G x y* **using** *that* **by** *simp*
then have *reachable x* **using** ‹*reachable y*›
unfolding *reachable-def* **by**(*rule converse-rtranclp-into-rtranclp*)
with ‹¬ *reachable x*› **show** *False* **by** *contradiction*
qed

have *reachable-V*: *vertex* Γ *x* **if** *reachable x* **for** *x*
proof −
from *that* **obtain** *p* **where** *p*: *path G x p a* **unfolding** *reachable-alt-def* **..**
then show *?thesis* **using** *rtrancl-path-nth*[*OF p*, *of 0*] *a*(*1*) *A-vertex*
by(*cases p* = [])(*auto 4 3 simp add*: *vertex-def elim*: *rtrancl-path.cases split*: *if-split-asm*)
qed

have *finite-j-IN*: ($\int^+$ *y*. *j* (*Some y*, *Some x*) ∂*count-space* (**IN** *x*)) ≠ ⊤ **for** *x*
proof −
have ($\int^+$ *y*. *j* (*Some y*, *Some x*) ∂*count-space* (**IN** *x*)) ≤ *d-IN f x*
by(*auto intro*!: *nn-integral-mono j-le-f simp add*: *d-IN-def nn-integral-count-space-indicator incoming-def split*: *split-indicator*)
thus *?thesis* **using** *currentD-finite-IN*[*OF f′*, *of x*] **by** (*auto simp*: *top-unique*)
qed
have *j-outside*: *j* (*x*, *y*) = *0* **if** ¬ *edge* Ψ *x y* **for** *x y*
using *that flowD-capacity*[*OF j*, *of* (*x*, *y*)] Ψ.*capacity-outside*[*of* (*x*, *y*)]
by(*auto*)

define *h* **where** *h* = (λ(*x*, *y*). *if reachable x* ∧ *reachable y then g* (*x*, *y*) *else 0*)
have *h-simps* [*simp*]: *h* (*x*, *y*) = (*if reachable x* ∧ *reachable y then g* (*x*, *y*) *else 0*) **for** *x y*
by(*simp add*: *h-def*)
have *h-le-g*: *h e* ≤ *g e* **for** *e* **by**(*cases e*) *simp*

have *OUT-h*: *d-OUT h x* = (*if reachable x then d-OUT g x else 0*) **for** *x*
proof −
have *d-OUT h x* = ($\sum^+$ *y*∈**OUT** *x*. *h* (*x*, *y*)) **using** *h-le-g currentD-outside*[*OF g*]
by(*intro d-OUT-alt-def*) *auto*
also have ... = ($\sum^+$ *y*∈**OUT** *x*. *if reachable x then g* (*x*, *y*) *else 0*)
by(*auto intro*!: *nn-integral-cong simp add*: *outgoing-def dest*: *reachable-AB*)
also have ... = (*if reachable x then d-OUT g x else 0*)
by(*auto intro*!: *d-OUT-alt-def*[*symmetric*] *currentD-outside*[*OF g*])
finally show *?thesis* **.**
qed
have *IN-h*: *d-IN h x* = (*if reachable x then d-IN g x else 0*) **for** *x*
proof −
have *d-IN h x* = ($\sum^+$ *y*∈**IN** *x*. *h* (*y*, *x*))
using *h-le-g currentD-outside*[*OF g*] **by**(*intro d-IN-alt-def*) *auto*
also have ... = ($\sum^+$ *y*∈**IN** *x*. *if reachable x then g* (*y*, *x*) *else 0*)
by(*auto intro*!: *nn-integral-cong simp add*: *incoming-def dest*: *reachable-BA*)
also have ... = (*if reachable x then d-IN g x else 0*)
by(*auto intro*!: *d-IN-alt-def*[*symmetric*] *currentD-outside*[*OF g*])
finally show *?thesis* **.**
qed

have *h*: *current* Γ *h* **using** *g h-le-g*
proof(*rule current-leI*)

show *d-OUT h x* $\leq$ *d-IN h x* **if** $x \notin A\ \Gamma$ **for** *x*
by(*simp add*: *OUT-h IN-h currentD-OUT-IN*[*OF g that*])
qed

have *reachable-full*: *j* (*None*, $\langle y \rangle$) = *u y* **if** *reach*: *reachable y* **for** *y*
proof(*rule ccontr*)
assume *j* (*None*, $\langle y \rangle$) $\neq$ *u y*
with *flowD-capacity*[*OF j*, *of* (*None*, $\langle y \rangle$)]
have *le*: *j* (*None*, $\langle y \rangle$) < *u y* **by**(*auto split*: *if-split-asm simp add*: *u-outside* Ψ.*flowD-outside*[*OF j*] *zero-less-iff-neq-zero*)
then obtain *y*: $y \in B\ \Gamma$ **and** *uy*: *u y* > *0* **using** *u-outside*[*of y*]
by(*cases* $y \in B\ \Gamma$; *cases u y* = *0*) (*auto simp add*: *zero-less-iff-neq-zero*)

from *reach* **obtain** *q* **where** *q*: *path G y q a* **and** *distinct*: *distinct* (*y* # *q*)
unfolding *reachable-alt-def* **by**(*auto elim*: *rtrancl-path-distinct*)
have *q-Nil*: *q* $\neq$ [] **using** *q a*(*1*) *disjoint y* **by**(*auto elim*!: *rtrancl-path.cases*)

let *?E* = *zip* (*y* # *q*) *q*
define *E* **where** *E* = (*None*, *Some y*) # *map* (*map-prod Some Some*) *?E*
define ζ **where** ζ = *Min* (*insert* (*u y* − *j* (*None*, *Some y*)) (*cap*$'$ ‘ *set ?E*))
let *?j*$'$ = λ*e*. (*if e* $\in$ *set E then* ζ *else 0*) + *j e*
define *j*$'$ **where** *j*$'$ = *cleanup ?j*$'$

have *j-free*: *0* < *cap*$'$ *e* **if** $e \in$ *set ?E* **for** *e* **using** *that* **unfolding** *E-def list.sel*
proof −
from *that* **obtain** *i* **where** *e*: *e* = ((*y* # *q*) ! *i*, *q* ! *i*)
and *i*: *i* < *length q* **by**(*auto simp add*: *set-zip*)
have *e*$'$: *edge G* ((*y* # *q*) ! *i*) (*q* ! *i*) **using** *q i* **by**(*rule rtrancl-path-nth*)
thus *?thesis* **using** *e* **by**(*simp*)
qed
have ζ*-pos*: *0* < ζ **unfolding** ζ*-def* **using** *le*
by(*auto intro*: *j-free diff-gr0-ennreal*)
have ζ*-le*: $\zeta \leq$ *cap*$'$ *e* **if** $e \in$ *set ?E* **for** *e* **using** *that* **unfolding** ζ*-def* **by** *auto*
have *finite-*ζ [*simplified*]: $\zeta < \top$ **unfolding** ζ*-def*
by(*intro Min-less-iff*[*THEN iffD2*])(*auto simp add*: *less-top*[*symmetric*])

have *E-antiparallel*: (x', y') $\in$ *set ?E* $\Longrightarrow$ (y', x') $\notin$ *set ?E* **for** x' y'
using *distinct*
apply(*auto simp add*: *in-set-zip nth-Cons in-set-conv-nth*)
apply(*auto simp add*: *distinct-conv-nth split*: *nat.split-asm*)
by (*metis Suc-lessD less-Suc-eq less-irrefl-nat*)

have *OUT-j*$'$: *d-OUT ?j*$'$ x' = ζ ∗ *card* (*set* [(x'', *y*) ← *E*. x'' = x']) + *d-OUT j* x' **for** x'
proof −
have *d-OUT ?j*$'$ x' = *d-OUT* (λ*e*. *if e* $\in$ *set E then* ζ *else 0*) x' + *d-OUT j* x'
using ζ*-pos* **by**(*intro d-OUT-add*)
also have *d-OUT* (λ*e*. *if e* $\in$ *set E then* ζ *else 0*) x' = $\int^{+}$ *y*. ζ ∗ *indicator* (*set E*) (x', *y*) ∂*count-space UNIV*

unfolding *d-OUT-def* **by**(*rule nn-integral-cong*)(*simp*)
also have $\ldots = (\int^+ e.\ \zeta * indicator\ (set\ E)\ e\ \partial embed\text{-}measure\ (count\text{-}space\ UNIV)\ (Pair\ x'))$
by(*simp add*: *measurable-embed-measure1 nn-integral-embed-measure*)
also have $\ldots = (\int^+ e.\ \zeta * indicator\ (set\ [(x'', y) \leftarrow E.\ x'' = x'])\ e\ \partial count\text{-}space\ UNIV)$
by(*auto simp add*: *embed-measure-count-space′ nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots = \zeta * card\ (set\ [(x'', y) \leftarrow E.\ x'' = x'])$ **using** *ζ-pos* **by**(*simp add*: *nn-integral-cmult*)
finally show *?thesis* **.**
qed
have *IN-j′*: $d\text{-}IN\ ?j'\ x' = \zeta * card\ (set\ [(y, x'') \leftarrow E.\ x'' = x']) + d\text{-}IN\ j\ x'$ **for** x'
proof −
have $d\text{-}IN\ ?j'\ x' = d\text{-}IN\ (\lambda e.\ if\ e \in set\ E\ then\ \zeta\ else\ 0)\ x' + d\text{-}IN\ j\ x'$
using *ζ-pos* **by**(*intro d-IN-add*)
also have $d\text{-}IN\ (\lambda e.\ if\ e \in set\ E\ then\ \zeta\ else\ 0)\ x' = \int^+ y.\ \zeta * indicator\ (set\ E)\ (y, x')\ \partial count\text{-}space\ UNIV$
unfolding *d-IN-def* **by**(*rule nn-integral-cong*)(*simp*)
also have $\ldots = (\int^+ e.\ \zeta * indicator\ (set\ E)\ e\ \partial embed\text{-}measure\ (count\text{-}space\ UNIV)\ (\lambda y.\ (y, x')))$
by(*simp add*: *measurable-embed-measure1 nn-integral-embed-measure*)
also have $\ldots = (\int^+ e.\ \zeta * indicator\ (set\ [(y, x'') \leftarrow E.\ x'' = x'])\ e\ \partial count\text{-}space\ UNIV)$
by(*auto simp add*: *embed-measure-count-space′ nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots = \zeta * card\ (set\ [(y, x'') \leftarrow E.\ x'' = x'])$
using *ζ-pos* **by**(*auto simp add*: *nn-integral-cmult*)
finally show *?thesis* **.**
qed

have *j′*: *flow* Ψ *j′*
proof
fix *e* :: *′v option edge*
consider (*None*) *e* = (*None*, *Some y*)
| (*Some*) *x y* **where** *e* = (*Some x*, *Some y*) (*x*, *y*) ∈ *set ?E*
| (*old*) *x y* **where** *e* = (*Some x*, *Some y*) (*x*, *y*) ∉ *set ?E*
| *y′* **where** *e* = (*None*, *Some y′*) *y* ≠ *y′*
| *e* = (*None*, *None*) | *x* **where** *e* = (*Some x*, *None*)
by(*cases e*; *case-tac a*; *case-tac b*)(*auto*)
then show $j'\ e \leq capacity\ \Psi\ e$ **using** *uy ζ-pos flowD-capacity*[*OF j*, *of e*]
proof(*cases*)
case *None*
have $\zeta \leq u\ y - j\ (None, Some\ y)$ **by**(*simp add*: *ζ-def*)
then have $\zeta + j\ (None, Some\ y) \leq u\ y$
using *ζ-pos* **by** (*auto simp add*: *ennreal-le-minus-iff*)
thus *?thesis* **using** *reachable-V*[*OF reach*] *None* Ψ.*flowD-outside*[*OF j*, *of* (*Some y*, *None*)] *uy*

by(*auto simp add*: *j′-def E-def*)
next
case (*Some x′ y′*)
have *e*: $\zeta \leq$ *cap′* (*x′*, *y′*) **using** *Some*(*2*) **by**(*rule ζ-le*)
then consider (*backward*) *edge* Γ *x′ y′ x′* ≠ *a* | (*forward*) *edge* Γ *y′ x′* ¬ *edge* Γ *x′ y′*
| (*a′*) *edge* Γ *x′ y′ x′* = *a*
using *Some ζ-pos* **by**(*auto split*: *if-split-asm*)
then show *?thesis*
proof *cases*
case *backward*
have $\zeta \leq f\ (x', y') + j\ (Some\ y', Some\ x') - j\ (Some\ x', Some\ y')$
using *e backward Some*(*1*) **by**(*simp add*: *g-simps*)
hence $\zeta + j\ (Some\ x', Some\ y') - j\ (Some\ y', Some\ x') \leq (f\ (x', y') + j\ (Some\ y', Some\ x') - j\ (Some\ x', Some\ y')) + j\ (Some\ x', Some\ y') - j\ (Some\ y', Some\ x')$
by(*intro ennreal-minus-mono add-right-mono*) *simp-all*
also have $\ldots = f\ (x', y')$
using *j-le-f*[*OF* ‹*edge* Γ *x′ y′*›]
by(*simp-all add*: *add-increasing2 less-top diff-add-assoc2-ennreal*)
finally show *?thesis* **using** *Some backward*
by(*auto simp add*: *j′-def E-def dest*: *in-set-tlD E-antiparallel*)
next
case *forward*
have $\zeta + j\ (Some\ x', Some\ y') - j\ (Some\ y', Some\ x') \leq \zeta + j\ (Some\ x', Some\ y')$
by(*rule diff-le-self-ennreal*)
also have $j\ (Some\ x', Some\ y') \leq$ *d-IN j* (*Some y′*)
by(*rule d-IN-ge-point*)
also have $\ldots \leq$ *weight* Γ *y′* **by**(*rule IN-j-le-weight*)
also have $\zeta \leq 1$ **using** *e forward* **by** *simp*
finally have $\zeta + j\ (Some\ x', Some\ y') - j\ (Some\ y', Some\ x') \leq max\ (weight\ \Gamma\ x')\ (weight\ \Gamma\ y') + 1$
by(*simp add*: *add-left-mono add-right-mono max-def*)(*metis* (*no-types*, *lifting*) *add.commute add-right-mono less-imp-le less-le-trans not-le*)
then show *?thesis* **using** *Some forward e*
by(*auto simp add*: *j′-def E-def max-def dest*: *in-set-tlD E-antiparallel*)
next
case *a′*
with *Some* **have** $a \in set\ (map\ fst\ (zip\ (y\ \#\ q)\ q))$ **by**(*auto intro*: *rev-image-eqI*)
also have $map\ fst\ (zip\ (y\ \#\ q)\ q) = butlast\ (y\ \#\ q)$ **by**(*induction q arbitrary*: *y*) *auto*
finally have *False* **using** *rtrancl-path-last*[*OF q q-Nil*] *distinct q-Nil*
by(*cases q rule*: *rev-cases*) *auto*
then show *?thesis* **..**
qed
next
case (*old x′ y′*)

hence $j'\ e \leq j\ e$ **using** *ζ-pos*
by(*auto simp add*: *j′-def E-def intro*!: *diff-le-self-ennreal*)
also have $j\ e \leq$ *capacity* $\Psi\ e$ **using** *j* **by**(*rule flowD-capacity*)
finally show *?thesis* **.**
qed(*auto simp add*: *j′-def E-def* Ψ.*flowD-outside*[*OF j*] *uy*)
next
fix x'
assume x': $x' \neq$ *source* Ψ $x' \neq$ *sink* Ψ
then obtain x'' **where** x'': $x' =$ *Some* x'' **by** *auto*
have *d-OUT ?j′* $x' = \zeta *$ *card* (*set* [$(x'', y) \leftarrow E.\ x'' = x'$]) + *d-OUT j* x' **by**(*rule OUT-j′*)
also have *card* (*set* [$(x'', y) \leftarrow E.\ x'' = x'$]) = *card* (*set* [$(y, x'') \leftarrow E.\ x'' = x'$]) (**is** *?lhs* = *?rhs*)
proof −
have *?lhs* = *length* [$(x'', y) \leftarrow E.\ x'' = x'$] **using** *distinct*
by(*subst distinct-card*)(*auto simp add*: *E-def filter-map distinct-map inj-map-prod′ distinct-zipI1*)
also have ... = *length* [$x''' \leftarrow$ *map fst ?E*. $x''' = x'$]
by(*simp add*: *E-def* x'' *split-beta cong*: *filter-cong*)
also have *map fst ?E* = *butlast* (y # q) **by**(*induction q arbitrary*: *y*) *simp-all*
also have [$x''' \leftarrow$ *butlast* (y # q). $x''' = x'$] = [$x''' \leftarrow y$ # q. $x''' = x'$]
using *q-Nil rtrancl-path-last*[*OF q q-Nil*] x' x''
by(*cases q rule*: *rev-cases*) *simp-all*
also have q = *map snd ?E* **by**(*induction q arbitrary*: *y*) *auto*
also have *length* [$x''' \leftarrow y$ # .... $x''' = x'$] = *length* [$x'' \leftarrow$ *map snd E*. $x'' = x'$] **using** x''
by(*simp add*: *E-def cong*: *filter-cong*)
also have ... = *length* [$(y, x'') \leftarrow E.\ x'' = x'$] **by**(*simp cong*: *filter-cong add*: *split-beta*)
also have ... = *?rhs* **using** *distinct*
by(*subst distinct-card*)(*auto simp add*: *E-def filter-map distinct-map inj-map-prod′ distinct-zipI1*)
finally show *?thesis* **.**
qed
also have $\zeta *$ ... + *d-OUT j* x' = *d-IN ?j′* x'
unfolding *flowD-KIR*[*OF j* x'] **by**(*rule IN-j′*[*symmetric*])
also have *d-IN ?j′* $x' \neq \top$
using Ψ.*flowD-finite-IN*[*OF j* x'(*2*)] *finite-ζ IN-j′*[*of* x'] **by** (*auto simp*: *top-add ennreal-mult-eq-top-iff*)
ultimately show *KIR* j' x' **unfolding** *j′-def* **by**(*rule KIR-cleanup*)
qed
hence *value-flow* Ψ $j' \leq \alpha$ **unfolding** *α-def* **by**(*auto intro*: *SUP-upper*)
moreover have *value-flow* Ψ j' > *value-flow* Ψ j
proof −
have *value-flow* Ψ j + *0* < *value-flow* Ψ $j + \zeta *$ *1*
using *ζ-pos value-j finite-flow* **by** *simp*
also have [$(x', y') \leftarrow E.\ x' =$ *None*] = [(*None*, *Some* y)]
using *q-Nil* **by**(*cases q*)(*auto simp add*: *E-def filter-map cong*: *filter-cong split-beta*)

hence $\zeta * 1 \leq \zeta * card$ (*set* [(x', y') ← *E*. x' = *None*]) **using** *ζ-pos*
by(*intro mult-left-mono*)(*auto simp add*: *E-def real-of-nat-ge-one-iff neq-Nil-conv card.insert-remove*)
also have *value-flow* Ψ *j* + ... = *value-flow* Ψ *?j'*
using *OUT-j'* **by**(*simp add*: *add.commute*)
also have ... = *value-flow* Ψ *j'* **unfolding** *j'-def*
by(*subst value-flow-cleanup*)(*auto simp add*: *E-def* Ψ.*flowD-outside*[*OF j*])
finally show *?thesis* **by**(*simp add*: *add-left-mono*)
qed
ultimately show *False* **using** *finite-flow ζ-pos value-j*
by(*cases value-flow* Ψ *j* ζ *rule*: *ennreal2-cases*) *simp-all*
qed

have *sep-h*: $y \in$ *TER h* **if** *reach*: *reachable y* **and** *y*: $y \in B\ \Gamma$ **and** *TER*: $y \in$ *?TER f* **for** *y*
proof(*rule ccontr*)
assume *y'*: $y \notin$ *TER h*
from *y a*(*1*) *disjoint* **have** *yna*: $y \neq a$ **by** *auto*

from *reach* **obtain** *p'* **where** *path G y p' a* **unfolding** *reachable-alt-def* **..**
then obtain *p'* **where** *p'*: *path G y p' a* **and** *distinct*: *distinct* (*y* # *p'*) **by**(*rule rtrancl-path-distinct*)

have *SINK*: $y \in$ *SINK h* **using** *y disjoint*
by(*auto simp add*: *SINK.simps d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0 intro*: *currentD-outside*[*OF g*] *dest*: *bipartite-E*)
have *hg*: *d-IN h y* = *d-IN g y* **using** *reach* **by**(*simp add*: *IN-h*)
also have ... = *d-IN f y* + *j* (*None*, *Some y*) **by**(*simp add*: *IN-g*)
also have *d-IN f y* = *weight* Γ *y* − *u y* **using** *currentD-weight-IN*[*OF f*, *of y*] *y disjoint TER*
by(*auto elim*!: *SAT.cases*)
also have *d-IN h y* < *weight* Γ *y* **using** *y' currentD-weight-IN*[*OF g*, *of y*] *y disjoint SINK*
by(*auto intro*: *SAT.intros*)
ultimately have *le*: *j* (*None*, *Some y*) < *u y*
by(*cases weight* Γ *y u y j* (*None*, *Some y*) *rule*: *ennreal3-cases*; *cases u y* ≤ *weight* Γ *y*)
(*auto simp*: *ennreal-minus ennreal-plus*[*symmetric*] *add-top ennreal-less-iff ennreal-neg simp del*: *ennreal-plus*)
moreover from *reach* **have** *j* (*None*, ⟨*y*⟩) = *u y* **by**(*rule reachable-full*)
ultimately show *False* **by** *simp*
qed

have *w'*: *wave* Γ *h*
proof
show *sep*: *separating* Γ (*TER h*)
proof(*rule ccontr*)
assume ¬ *?thesis*
then obtain *x p y* **where** *x*: $x \in A\ \Gamma$ **and** *y*: $y \in B\ \Gamma$ **and** *p*: *path* Γ *x p y*

and x': $x \notin$ *TER h* **and** *bypass*: $\bigwedge z.\ z \in$ *set p* $\Longrightarrow z \notin$ *TER h*
by(*auto simp add*: *separating-gen.simps*)
from *p disjoint x y* **have** *p-eq*: $p = [y]$ **and** *edge*: $(x,\ y) \in$ **E**
by $-$(*erule rtrancl-path.cases*, *auto dest*: *bipartite-E*)+
from *p-eq bypass* **have** y': $y \notin$ *TER h* **by** *simp*
have *reachable x* **using** x' **by**(*rule contrapos-np*)(*simp add*: *SINK.simps d-OUT-def SAT.A x*)
hence *reach*: *reachable y* **using** *edge* **by**(*rule reachable-AB*)

have $*$: $x \notin \mathcal{E}_{?\Gamma}$ (*?TER f*) **using** x'
proof(*rule contrapos-nn*)
assume $*$: $x \in \mathcal{E}_{?\Gamma}$ (*?TER f*)
have *d-OUT h x* $\leq$ *d-OUT g x* **using** *h-le-g* **by**(*rule d-OUT-mono*)
also from $*$ **have** $x \neq a$ **using** *a* **by** *auto*
then have *d-OUT j* (*Some x*) $=$ *d-IN j* (*Some x*) **by**(*auto intro*: *flowD-KIR*[*OF j*])
hence *d-OUT g x* $\leq$ *d-OUT f x* **using** *OUT-g-A*[*OF x*] *IN-j*[*of Some x*] *finite-flow*
by(*auto split*: *if-split-asm*)
also have $\ldots = 0$ **using** $*$ **by**(*auto elim*: *SINK.cases*)
finally have $x \in$ *SINK h* **by**(*simp add*: *SINK.simps*)
with *x* **show** $x \in$ *TER h* **by**(*simp add*: *SAT.A*)
qed
from *p p-eq x y* **have** *path* $?\Gamma$ *x* $[y]$ *y* $x \in A$ $?\Gamma$ $y \in B$ $?\Gamma$ **by** *simp-all*
from $*$ *separatingD*[*OF separating-essential*, *OF waveD-separating*, *OF w this*]
have $y \in$ *?TER f* **by** *auto*
with *reach y* **have** $y \in$ *TER h* **by**(*rule sep-h*)
with y' **show** *False* **by** *contradiction*
qed
qed(*rule h*)

have *OUT-g-a*: *d-OUT g a* $=$ *d-OUT h a* **by**(*simp add*: *OUT-h*)
have $a \notin \mathcal{E}$ (*TER h*)
proof
assume $*$: $a \in \mathcal{E}$ (*TER h*)

have *j* (*Some a*, *Some y*) $= 0$ **for** *y*
using *flowD-capacity*[*OF j*, *of* (*Some a*, *Some y*)] *a*(*1*) *disjoint*
by(*auto split*: *if-split-asm dest*: *bipartite-E*)
then have *d-OUT f a* $\leq$ *d-OUT g a* **unfolding** *d-OUT-def*
— This step requires that *j* does not decrease the outflow of *a*. That's why we set the capacity of the outgoing edges from $\langle a \rangle$ in Ψ to *0*
by(*intro nn-integral-mono*)(*auto simp add*: *g-simps currentD-outside*[*OF f*] *intro*:)
then have $a \in$ *SINK f* **using** *OUT-g-a* $*$ **by**(*simp add*: *SINK.simps*)
with *a*(*1*) **have** $a \in$ *?TER f* **by**(*auto intro*: *SAT.A*)
with *a*(*2*) **have** a': $\neg$ *essential* Γ ($B\ \Gamma$) (*?TER f*) *a* **by** *simp*

from $*$ **obtain** *y* **where** *ay*: *edge* Γ *a y* **and** *y*: $y \in B\ \Gamma$ **and** y': $y \notin$ *TER h*

```
using disjoint a(1)
     by(auto 4 4 simp add: essential-def elim: rtrancl-path.cases dest: bipartite-E)
   from not-essentialD[OF a' rtrancl-path.step, OF ay rtrancl-path.base y]
   have TER: y ∈ ?TER f by simp

   have reachable y using ‹reachable a› by(rule reachable-AB)(simp add: ay)
   hence y ∈ TER h using y TER by(rule sep-h)
   with y' show False by contradiction
  qed
  with ‹a ∈ A Γ› have hindrance Γ h
  proof
   have d-OUT h a = d-OUT g a by(simp add: OUT-g-a)
   also have ... ≤ d-OUT f a + ∫+ y. j (Some y, Some a) ∂count-space UNIV
     unfolding d-OUT-def d-IN-def
    by(subst nn-integral-add[symmetric])(auto simp add: g-simps intro!: nn-integral-mono
diff-le-self-ennreal)
    also have (∫+ y. j (Some y, Some a) ∂count-space UNIV) = (∫+ y. j (y,
Some a) ∂embed-measure (count-space UNIV) Some)
     by(simp add: nn-integral-embed-measure measurable-embed-measure1)
   also have ... ≤ d-IN j (Some a) unfolding d-IN-def
    by(auto simp add: embed-measure-count-space nn-integral-count-space-indicator
intro!: nn-integral-mono split: split-indicator)
   also have ... ≤ α by(rule IN-j)
   also have ... ≤ ε by(rule α-le)
   also have d-OUT f a + ... < d-OUT f a + (weight Γ a − d-OUT f a) using
ε-less
     using currentD-finite-OUT[OF f'] by (simp add: ennreal-add-left-cancel-less)
   also have ... = weight Γ a
     using a-le by simp
   finally show d-OUT h a < weight Γ a by(simp add: add-left-mono)
  qed
  then show ?thesis using h w' by(blast intro: hindered.intros)
qed

end

corollary hindered-reduce-current: — Corollary 6.8
  fixes ε g
  defines ε ≡ ∑+ x∈B Γ. d-IN g x − d-OUT g x
  assumes g: current Γ g
  and ε-finite: ε ≠ ⊤
  and hindered: hindered-by (Γ ⊖ g) ε
  shows hindered Γ
proof −
  define Γ' where Γ' = Γ(|weight := λx. if x ∈ A Γ then weight Γ x − d-OUT g
x else weight Γ x|)
  have Γ'-sel [simp]:
   edge Γ' = edge Γ
   A Γ' = A Γ
```

$B\ \Gamma' = B\ \Gamma$
weight Γ' *x* = (*if x* ∈ *A* Γ *then weight* Γ *x* − *d-OUT g x else weight* Γ *x*)
vertex Γ' = *vertex* Γ
web.more Γ' = *web.more* Γ
for *x* **by**(*simp-all add*: Γ'*-def*)
have *countable-bipartite-web* Γ'
by *unfold-locales*(*simp-all add*: *A-in B-out A-vertex disjoint bipartite-V no-loop weight-outside currentD-outside-OUT*[*OF g*] *currentD-weight-OUT*[*OF g*] *edge-antiparallel*, *rule bipartite-E*)
then interpret Γ': *countable-bipartite-web* Γ' **.**
let *?u* = λ*x*. (*d-IN g x* − *d-OUT g x*) ∗ *indicator* (− *A* Γ) *x*

have *hindered* Γ'
proof(*rule* Γ'*.hindered-reduce*)
show *?u x* = *0* **if** *x* ∉ *B* Γ' **for** *x* **using** *that bipartite-V*
by(*cases vertex* Γ' *x*)(*auto simp add*: *currentD-outside-OUT*[*OF g*] *currentD-outside-IN*[*OF g*])

have ∗: ($\sum^+$ *x*∈*B* Γ'. *?u x*) = ε **using** *disjoint*
by(*auto intro*!: *nn-integral-cong simp add*: *ε-def nn-integral-count-space-indicator currentD-outside-OUT*[*OF g*] *currentD-outside-IN*[*OF g*] *not-vertex split*: *split-indicator*)
thus ($\sum^+$ *x*∈*B* Γ'. *?u x*) ≠ ⊤ **using** *ε-finite* **by** *simp*

have ∗∗: Γ'⦇*weight* := *weight* Γ' − *?u*⦈ = Γ ⊖ *g*
using *currentD-weight-IN*[*OF g*] *currentD-OUT-IN*[*OF g*] *currentD-IN*[*OF g*] *currentD-finite-OUT*[*OF g*]
by(*intro web.equality*)(*simp-all add*: *fun-eq-iff diff-diff-ennreal′ ennreal-diff-le-mono-left*)
show *hindered-by* (Γ'⦇*weight* := *weight* Γ' − *?u*⦈) ($\sum^+$ *x*∈*B* Γ'. *?u x*)
unfolding ∗ ∗∗ **by**(*fact hindered*)
show (λ*x*. (*d-IN g x* − *d-OUT g x*) ∗ *indicator* (− *A* Γ) *x*) ≤ *weight* Γ'
using *currentD-weight-IN*[*OF g*]
by (*simp add*: *le-fun-def ennreal-diff-le-mono-left*)
qed
then show *?thesis*
by(*rule hindered-mono-web*[*rotated* −*1*]) *simp-all*
qed

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
by(*simp-all add*: *reduce-weight-def zero-ennreal-def*[*symmetric*] *vertex-def fun-eq-iff*)

lemma *essential-reduce-weight* [*simp*]: *essential* (*reduce-weight* Γ x r) = *essential* Γ
by(*simp add*: *fun-eq-iff essential-def*)

lemma *roofed-reduce-weight* [*simp*]: *roofed-gen* (*reduce-weight* Γ x r) = *roofed-gen* Γ
by(*simp add*: *fun-eq-iff roofed-def*)

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*: $x \notin$ *range Inner* $\longleftrightarrow$ x = *SOURCE* $\vee$ x = *SINK*
by(*cases* x) *auto*

private lemma *inj-Inner* [*simp*]: $\bigwedge A$. *inj-on Inner* A
by(*simp add*: *inj-on-def*)

lemma *unhinder-bipartite*:
  assumes h: $\bigwedge n$ :: *nat*. *current* Γ (h n)
  and *SAT*: $\bigwedge n$. (B Γ $\cap$ **V**) $-$ $\{b\}$ $\subseteq$ *SAT* Γ (h n)
  and b: $b \in B$ 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*: $b \in$ **V**
  shows $\exists h'$. *current* Γ h' $\wedge$ *wave* Γ h' $\wedge$ B Γ $\cap$ **V** $\subseteq$ *SAT* Γ h'
proof $-$
  write *Inner* (‹⟨-⟩›)
  define *edge'*
    where *edge' xo yo* =
      (*case* (*xo*, *yo*) *of*
        (⟨x⟩, ⟨y⟩) $\Rightarrow$ *edge* Γ x y $\vee$ *edge* Γ y x
      | (⟨x⟩, *SINK*) $\Rightarrow$ $x \in A$ Γ
      | (*SOURCE*, ⟨y⟩) $\Rightarrow$ $y = b$
      | (*SINK*, ⟨x⟩) $\Rightarrow$ $x \in A$ Γ
      | - $\Rightarrow$ *False*) **for** *xo yo*
  have *edge'-simps* [*simp*]:
    edge' ⟨x⟩ ⟨y⟩ $\longleftrightarrow$ *edge* Γ x y $\vee$ *edge* Γ y x
    edge' ⟨x⟩ *SINK* $\longleftrightarrow$ $x \in A$ Γ
    edge' SOURCE yo $\longleftrightarrow$ *yo* = ⟨b⟩

$edge'$ $SINK$ $\langle x\rangle \longleftrightarrow x \in A\ \Gamma$
$edge'$ $SINK$ $SINK \longleftrightarrow False$
$edge'$ xo $SOURCE \longleftrightarrow False$
for $x\ y\ yo\ xo$ **by**(*simp-all add*: *edge′-def split*: *vertex.split*)
have *edge′E*: *thesis* **if** *edge′ xo yo*
$\bigwedge x\ y.\ [\![\ xo = \langle x\rangle;\ yo = \langle y\rangle;\ edge\ \Gamma\ x\ y \vee edge\ \Gamma\ y\ x\ ]\!] \Longrightarrow thesis$
$\bigwedge x.\ [\![\ xo = \langle x\rangle;\ yo = SINK;\ x \in A\ \Gamma\ ]\!] \Longrightarrow thesis$
$\bigwedge x.\ [\![\ xo = SOURCE;\ yo = \langle b\rangle\ ]\!] \Longrightarrow thesis$
$\bigwedge y.\ [\![\ xo = SINK;\ yo = \langle y\rangle;\ y \in A\ \Gamma\ ]\!] \Longrightarrow thesis$
for *xo yo thesis* **using** *that* **by**(*auto simp add*: *edge′-def split*: *option.split-asm vertex.split-asm*)
have *edge′-Inner1* [*elim*!]: *thesis* **if** *edge′* $\langle x\rangle$ *yo*
$\bigwedge y.\ [\![\ yo = \langle y\rangle;\ edge\ \Gamma\ x\ y \vee edge\ \Gamma\ y\ x\ ]\!] \Longrightarrow thesis$
$[\![\ yo = SINK;\ x \in A\ \Gamma\ ]\!] \Longrightarrow thesis$
for *x yo thesis* **using** *that* **by**(*auto elim*: *edge′E*)
have *edge′-Inner2* [*elim*!]: *thesis* **if** *edge′ xo* $\langle y\rangle$
$\bigwedge x.\ [\![\ xo = \langle x\rangle;\ edge\ \Gamma\ x\ y \vee edge\ \Gamma\ y\ x\ ]\!] \Longrightarrow thesis$
$[\![\ xo = SOURCE;\ y = b\ ]\!] \Longrightarrow thesis$
$[\![\ xo = SINK;\ y \in A\ \Gamma\ ]\!] \Longrightarrow thesis$
for *xo y thesis* **using** *that* **by**(*auto elim*: *edge′E*)
have *edge′-SINK1* [*elim*!]: *thesis* **if** *edge′ SINK yo*
$\bigwedge y.\ [\![\ yo = \langle y\rangle;\ y \in A\ \Gamma\ ]\!] \Longrightarrow thesis$
for *yo thesis* **using** *that* **by**(*auto elim*: *edge′E*)
have *edge′-SINK2* [*elim*!]: *thesis* **if** *edge′ xo SINK*
$\bigwedge x.\ [\![\ xo = \langle x\rangle;\ x \in A\ \Gamma\ ]\!] \Longrightarrow thesis$
for *xo thesis* **using** *that* **by**(*auto elim*: *edge′E*)

define *cap*
where *cap xoyo* =
(*case xoyo of*
$(\langle x\rangle, \langle y\rangle) \Rightarrow$ *if edge* Γ *x y then h 0* (x, y) *else if edge* Γ *y x then max* (*weight* Γ *x*) (*weight* Γ *y*) *else 0*
| $(\langle x\rangle, SINK) \Rightarrow$ *if* $x \in A\ \Gamma$ *then weight* $\Gamma\ x -$ *d-OUT* (*h 0*) *x else 0*
| $(SOURCE, yo) \Rightarrow$ *if* $yo = \langle b\rangle$ *then weight* $\Gamma\ b -$ *d-IN* (*h 0*) *b else 0*
| $(SINK, \langle y\rangle) \Rightarrow$ *if* $y \in A\ \Gamma$ *then weight* Γ *y else 0*
| - $\Rightarrow$ *0*) **for** *xoyo*
have *cap-simps* [*simp*]:
cap $(\langle x\rangle, \langle y\rangle) =$ (*if edge* Γ *x y then h 0* (x, y) *else if edge* Γ *y x then max* (*weight* Γ *x*) (*weight* Γ *y*) *else 0*)
cap $(\langle x\rangle, SINK) =$ (*if* $x \in A\ \Gamma$ *then weight* $\Gamma\ x -$ *d-OUT* (*h 0*) *x else 0*)
cap $(SOURCE, yo) =$ (*if* $yo = \langle b\rangle$ *then weight* $\Gamma\ b -$ *d-IN* (*h 0*) *b else 0*)
cap $(SINK, \langle y\rangle) =$ (*if* $y \in A\ \Gamma$ *then weight* Γ *y else 0*)
cap $(SINK, SINK) = 0$
cap $(xo, SOURCE) = 0$
for *x y yo xo* **by**(*simp-all add*: *cap-def split*: *vertex.split*)
define Ψ **where** $\Psi = (\!|edge = edge',\ capacity = cap,\ source = SOURCE,\ sink = SINK|\!)$
have Ψ*-sel* [*simp*]:
edge $\Psi = edge'$

capacity Ψ = *cap*
source Ψ = *SOURCE*
sink Ψ = *SINK*
by(*simp-all add*: Ψ-*def*)

have *cap-outside1*: $\neg$ *vertex* Γ *x* $\Longrightarrow$ *cap* ($\langle x\rangle$, *y*) = *0* **for** *x y* **using** *A-vertex*
by(*cases y*)(*auto simp add*: *vertex-def*)
have *capacity-A-weight*: *d-OUT cap* $\langle x\rangle$ $\leq$ *2* $*$ *weight* Γ *x* **if** *x* $\in$ *A* Γ **for** *x*
proof −
have *d-OUT cap* $\langle x\rangle$ $\leq$ ($\sum^+$ *y*. *h 0* (*x*, *inner y*) $*$ *indicator* (*range Inner*) *y* + *weight* Γ *x* $*$ *indicator* {*SINK*} *y*)
using *that disjoint* **unfolding** *d-OUT-def*
by(*auto intro*!: *nn-integral-mono diff-le-self-ennreal simp add*: *A-in notin-range-Inner split*: *split-indicator*)
also have ... = ($\sum^+$ *y*$\in$*range Inner*. *h 0* (*x*, *inner y*)) + *weight* Γ *x*
by(*auto simp add*: *nn-integral-count-space-indicator nn-integral-add*)
also have ($\sum^+$ *y*$\in$*range Inner*. *h 0* (*x*, *inner y*)) = *d-OUT* (*h 0*) *x*
by(*simp add*: *d-OUT-def nn-integral-count-space-reindex*)
also have ... $\leq$ *weight* Γ *x* **using** *h* **by**(*rule currentD-weight-OUT*)
finally show *?thesis* **unfolding** *one-add-one*[*symmetric*] *distrib-right* **by**(*simp add*: *add-right-mono*)
qed
have *flow-attainability*: *flow-attainability* Ψ
proof
have $\mathbf{E}_\Psi$ $\subseteq$ (λ(*x*, *y*). ($\langle x\rangle$, $\langle y\rangle$)) ‘ **E** $\cup$ (λ(*x*, *y*). ($\langle y\rangle$, $\langle x\rangle$)) ‘ **E** $\cup$ (λ*x*. ($\langle x\rangle$, *SINK*)) ‘ *A* Γ $\cup$ (λ*x*. (*SINK*, $\langle x\rangle$)) ‘ *A* Γ $\cup$ {(*SOURCE*, $\langle b\rangle$)}
by(*auto simp add*: *edge'-def split*: *vertex.split-asm*)
moreover have *countable* (*A* Γ) **using** *A-vertex* **by**(*rule countable-subset*) *simp*
ultimately show *countable* $\mathbf{E}_\Psi$ **by**(*auto elim*: *countable-subset*)
next
fix *v*
assume *v* $\neq$ *sink* Ψ
then consider (*source*) *v* = *SOURCE* | (*A*) *x* **where** *v* = $\langle x\rangle$ *x* $\in$ *A* Γ
| (*B*) *y* **where** *v* = $\langle y\rangle$ *y* $\notin$ *A* Γ *y* $\in$ **V** | (*outside*) *x* **where** *v* = $\langle x\rangle$ *x* $\notin$ **V**
by(*cases v*) *auto*
then show *d-IN* (*capacity* Ψ) *v* $\neq$ $\top$ $\vee$ *d-OUT* (*capacity* Ψ) *v* $\neq$ $\top$
proof *cases*
case *source* **thus** *?thesis* **by**(*simp add*: *d-IN-def*)
next
case (*A x*)
thus *?thesis* **using** *capacity-A-weight*[*of x*] **by** (*auto simp*: *top-unique ennreal-mult-eq-top-iff*)
next
case (*B y*)
have *d-IN* (*capacity* Ψ) *v* $\leq$ ($\sum^+$ *x*. *h 0* (*inner x*, *y*) $*$ *indicator* (*range Inner*) *x* + *weight* Γ *b* $*$ *indicator* {*SOURCE*} *x*)
using *B bipartite-V*
by(*auto 4 4 intro*!: *nn-integral-mono simp add*: *diff-le-self-ennreal d-IN-def notin-range-Inner nn-integral-count-space-indicator currentD-outside*[*OF h*] *split*:

split-indicator dest: *bipartite-E*)
also have ... $=$ ($\sum^+$ *x*∈*range Inner*. *h 0* (*inner x*, *y*)) + *weight* Γ *b*
by(*simp add*: *nn-integral-add nn-integral-count-space-indicator*)
also have ($\sum^+$ *x*∈*range Inner*. *h 0* (*inner x*, *y*)) = *d-IN* (*h 0*) *y*
by(*simp add*: *d-IN-def nn-integral-count-space-reindex*)
also have *d-IN* (*h 0*) *y* ≤ *weight* Γ *y* **using** *h* **by**(*rule currentD-weight-IN*)
finally show *?thesis* **by**(*auto simp add*: *top-unique add-right-mono split*: *if-split-asm*)
next
case *outside*
hence *d-OUT* (*capacity* Ψ) *v* = *0* **using** *A-vertex*
by(*auto simp add*: *d-OUT-def nn-integral-0-iff-AE AE-count-space cap-def vertex-def split*: *vertex.split*)
thus *?thesis* **by** *simp*
qed
next
show *capacity* Ψ *e* ≠ ⊤ **for** *e*
by(*auto simp add*: *cap-def max-def vertex-def currentD-finite*[*OF h*] *split*: *vertex.split prod.split*)
show *capacity* Ψ *e* = *0* **if** *e* ∉ $\mathbf{E}_\Psi$ **for** *e* **using** *that*
by(*auto simp add*: *cap-def max-def split*: *prod.split*; *split vertex.split*)+
show ¬ *edge* Ψ *x* (*source* Ψ) **for** *x* **using** *b* **by**(*auto simp add*: *B-out*)
show ¬ *edge* Ψ *x x* **for** *x* **by**(*cases x*)(*simp-all add*: *no-loop*)
show *source* Ψ ≠ *sink* Ψ **by** *simp*
qed
then interpret Ψ: *flow-attainability* Ψ **.**
define α **where** α = (*SUP f*∈{*f*. *flow* Ψ *f*}. *value-flow* Ψ *f*)

define *f*
where *f n xoyo* =
(*case xoyo of*
(⟨*x*⟩, ⟨*y*⟩) ⇒ *if edge* Γ *x y then h 0* (*x*, *y*) − *h n* (*x*, *y*) *else if edge* Γ *y x then h n* (*y*, *x*) − *h 0* (*y*, *x*) *else 0*
| (*SOURCE*, ⟨*y*⟩) ⇒ *if y* = *b then d-IN* (*h n*) *b* − *d-IN* (*h 0*) *b else 0*
| (⟨*x*⟩, *SINK*) ⇒ *if x* ∈ *A* Γ *then d-OUT* (*h n*) *x* − *d-OUT* (*h 0*) *x else 0*
| (*SINK*, ⟨*y*⟩) ⇒ *if y* ∈ *A* Γ *then d-OUT* (*h 0*) *y* − *d-OUT* (*h n*) *y else 0*
| - ⇒ *0*) **for** *n xoyo*
have *f-cases*: *thesis* **if** ⋀*x y*. *e* = (⟨*x*⟩, ⟨*y*⟩) ⟹ *thesis* ⋀*y*. *e* = (*SOURCE*, ⟨*y*⟩) ⟹ *thesis*
⋀*x*. *e* = (⟨*x*⟩, *SINK*) ⟹ *thesis* ⋀*y*. *e* = (*SINK*, ⟨*y*⟩) ⟹ *thesis e* = (*SINK*, *SINK*) ⟹ *thesis*
⋀*xo*. *e* = (*xo*, *SOURCE*) ⟹ *thesis e* = (*SOURCE*, *SINK*) ⟹ *thesis*
for *e* :: *'v vertex edge* **and** *thesis*
using *that* **by**(*cases e*; *cases fst e snd e rule*: *vertex.exhaust*[*case-product vertex.exhaust*]) *simp-all*
have *f-simps* [*simp*]:
f n (⟨*x*⟩, ⟨*y*⟩) = (*if edge* Γ *x y then h 0* (*x*, *y*) − *h n* (*x*, *y*) *else if edge* Γ *y x then h n* (*y*, *x*) − *h 0* (*y*, *x*) *else 0*)
f n (*SOURCE*, ⟨*y*⟩) = (*if y* = *b then d-IN* (*h n*) *b* − *d-IN* (*h 0*) *b else 0*)

f n (⟨*x*⟩, *SINK*) = (*if x* ∈ *A* Γ *then d-OUT* (*h n*) *x* − *d-OUT* (*h 0*) *x else 0*)
f n (*SINK*, ⟨*y*⟩) = (*if y* ∈ *A* Γ *then d-OUT* (*h 0*) *y* − *d-OUT* (*h n*) *y else 0*)
f n (*SOURCE*, *SINK*) = *0*
f n (*SINK*, *SINK*) = *0*
f n (*xo*, *SOURCE*) = *0*
for *n x y xo* **by**(*simp-all add*: *f-def split*: *vertex.split*)
have *OUT-f-SOURCE*: *d-OUT* (*f n*) *SOURCE* = *d-IN* (*h n*) *b* − *d-IN* (*h 0*) *b*
for *n*
proof(*rule trans*)
show *d-OUT* (*f n*) *SOURCE* = ($\sum^+$ *y*. *f n* (*SOURCE*, *y*) ∗ *indicator* {⟨*b*⟩} *y*) **unfolding** *d-OUT-def*
apply(*rule nn-integral-cong*) **subgoal for** *x* **by**(*cases x*) *auto* **done**
show . . . = *d-IN* (*h n*) *b* − *d-IN* (*h 0*) *b* **using** *h0-b*[*of n*]
by(*auto simp add*: *max-def*)
qed

have *OUT-f-outside*: *d-OUT* (*f n*) ⟨*x*⟩ = *0* **if** *x* ∉ **V** **for** *x n* **using** *A-vertex that*
apply(*clarsimp simp add*: *d-OUT-def nn-integral-0-iff emeasure-count-space-eq-0*)
subgoal for *y* **by**(*cases y*)(*auto simp add*: *vertex-def*)
done
have *IN-f-outside*: *d-IN* (*f n*) ⟨*x*⟩ = *0* **if** *x* ∉ **V** **for** *x n* **using** *b-V that*
apply(*clarsimp simp add*: *d-IN-def nn-integral-0-iff emeasure-count-space-eq-0*)
subgoal for *y* **by**(*cases y*)(*auto simp add*: *currentD-outside-OUT*[*OF h*] *vertex-def*)
done

have *f*: *flow* Ψ (*f n*) **for** *n*
proof
show *f-le*: *f n e* ≤ *capacity* Ψ *e* **for** *e*
using *currentD-weight-out*[*OF h*] *currentD-weight-IN*[*OF h*] *currentD-weight-OUT*[*OF h*]
by(*cases e rule*: *f-cases*)
(*auto dest*: *edge-antiparallel simp add*: *not-le le-max-iff-disj intro*: *ennreal-minus-mono ennreal-diff-le-mono-left*)

fix *xo*
assume *xo* ≠ *source* Ψ *xo* ≠ *sink* Ψ
then consider (*A*) *x* **where** *xo* = ⟨*x*⟩ *x* ∈ *A* Γ | (*B*) *x* **where** *xo* = ⟨*x*⟩ *x* ∈ *B* Γ *x* ∈ **V**
| (*outside*) *x* **where** *xo* = ⟨*x*⟩ *x* ∉ **V** **using** *bipartite-V* **by**(*cases xo*) *auto*
then show *KIR* (*f n*) *xo*
proof *cases*
case *outside*
thus *?thesis* **by**(*simp add*: *OUT-f-outside IN-f-outside*)
next
case *A*

have *finite1*: ($\sum^+$ *y*. *h n* (*x*, *y*) ∗ *indicator A y*) ≠ ⊤ **for** *A n*
using *currentD-finite-OUT*[*OF h*, *of n x*, *unfolded d-OUT-def*]

by(*rule neq-top-trans*)(*auto intro*!: *nn-integral-mono simp add*: *split*: *split-indicator*)

let *?h0-ge-hn* = {*y*. *h 0* (*x*, *y*) ≥ *h n* (*x*, *y*)}
let *?h0-lt-hn* = {*y*. *h 0* (*x*, *y*) < *h n* (*x*, *y*)}

have *d-OUT* (*f n*) ⟨*x*⟩ = (∑⁺ *y*. *f n* (⟨*x*⟩, *y*) ∗ *indicator* (*range Inner*) *y* + *f n* (⟨*x*⟩, *y*) ∗ *indicator* {*SINK*} *y*)
unfolding *d-OUT-def* **by**(*intro nn-integral-cong*)(*auto split*: *split-indicator simp add*: *notin-range-Inner*)
also have . . . = (∑⁺ *y*∈*range Inner*. *f n* (⟨*x*⟩, *y*)) + *f n* (⟨*x*⟩, *SINK*)
by(*simp add*: *nn-integral-add nn-integral-count-space-indicator max.left-commute max.commute*)
also have (∑⁺ *y*∈*range Inner*. *f n* (⟨*x*⟩, *y*)) = (∑⁺ *y*. *h 0* (*x*, *y*) − *h n* (*x*, *y*)) **using** *A*
apply(*simp add*: *nn-integral-count-space-reindex cong*: *nn-integral-cong-simp outgoing-def*)
apply(*auto simp add*: *nn-integral-count-space-indicator outgoing-def A-in max.absorb1 currentD-outside*[*OF h*] *intro*!: *nn-integral-cong split*: *split-indicator dest*: *edge-antiparallel*)
done
also have . . . = (∑⁺ *y*. *h 0* (*x*, *y*) ∗ *indicator ?h0-ge-hn y*) − (∑⁺ *y*. *h n* (*x*, *y*) ∗ *indicator ?h0-ge-hn y*)
apply(*subst nn-integral-diff*[*symmetric*])
apply(*simp-all add*: *AE-count-space finite1 split*: *split-indicator*)
apply(*rule nn-integral-cong*; *auto simp add*: *max-def not-le split*: *split-indicator*)
by (*metis diff-eq-0-ennreal le-less not-le top-greatest*)
also have (∑⁺ *y*. *h n* (*x*, *y*) ∗ *indicator ?h0-ge-hn y*) = *d-OUT* (*h n*) *x* − (∑⁺ *y*. *h n* (*x*, *y*) ∗ *indicator ?h0-lt-hn y*)
unfolding *d-OUT-def*
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space finite1 currentD-finite*[*OF h*] *split*: *split-indicator intro*!: *nn-integral-cong*)
done
also have (∑⁺ *y*. *h 0* (*x*, *y*) ∗ *indicator ?h0-ge-hn y*) − . . . + *f n* (⟨*x*⟩, *SINK*) =
(∑⁺ *y*. *h 0* (*x*, *y*) ∗ *indicator ?h0-ge-hn y*) + (∑⁺ *y*. *h n* (*x*, *y*) ∗ *indicator ?h0-lt-hn y*) − *min* (*d-OUT* (*h n*) *x*) (*d-OUT* (*h 0*) *x*)
using *finite1*[*of n* {-}] *A finite1*[*of n UNIV*]
apply (*subst diff-diff-ennreal′*)
apply (*auto simp*: *d-OUT-def finite1 AE-count-space nn-integral-diff*[*symmetric*] *top-unique nn-integral-add*[*symmetric*]
split: *split-indicator intro*!: *nn-integral-mono ennreal-diff-self*)
apply (*simp add*: *min-def not-le diff-eq-0-ennreal finite1 less-top*[*symmetric*])
apply (*subst diff-add-assoc2-ennreal*)
apply (*auto simp*: *add-diff-eq-ennreal intro*!: *nn-integral-mono split*: *split-indicator*)
apply (*subst diff-diff-commute-ennreal*)
apply (*simp add*: *ennreal-add-diff-cancel*)
done
also have . . . = (∑⁺ *y*. *h n* (*x*, *y*) ∗ *indicator ?h0-lt-hn y*) − (*d-OUT* (*h 0*)

$x - (\sum^{+} y.\ h\ 0\ (x,\ y) * indicator\ ?h0\text{-}ge\text{-}hn\ y)) + f\ n\ (SINK,\ \langle x\rangle)$
apply(*rule sym*)
using *finite1*[*of 0* {-}] *A finite1*[*of 0 UNIV*]
apply (*subst diff-diff-ennreal′*)
apply (*auto simp*: *d-OUT-def finite1 AE-count-space nn-integral-diff*[*symmetric*] *top-unique nn-integral-add*[*symmetric*]
split: *split-indicator intro*!: *nn-integral-mono ennreal-diff-self*)
apply (*simp add*: *min-def not-le diff-eq-0-ennreal finite1 less-top*[*symmetric*])
apply (*subst diff-add-assoc2-ennreal*)
apply (*auto simp*: *add-diff-eq-ennreal intro*!: *nn-integral-mono split*: *split-indicator*)
apply (*subst diff-diff-commute-ennreal*)
apply (*simp-all add*: *ennreal-add-diff-cancel ac-simps*)
done
also have $d\text{-}OUT\ (h\ 0)\ x - (\sum^{+} y.\ h\ 0\ (x,\ y) * indicator\ ?h0\text{-}ge\text{-}hn\ y) = (\sum^{+} y.\ h\ 0\ (x,\ y) * indicator\ ?h0\text{-}lt\text{-}hn\ y)$
unfolding *d-OUT-def*
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space finite1 currentD-finite*[*OF h*] *split*: *split-indicator intro*!: *nn-integral-cong*)
done
also have $(\sum^{+} y.\ h\ n\ (x,\ y) * indicator\ ?h0\text{-}lt\text{-}hn\ y) - \ldots = (\sum^{+} y.\ h\ n\ (x,\ y) - h\ 0\ (x,\ y))$
apply(*subst nn-integral-diff*[*symmetric*])
apply(*simp-all add*: *AE-count-space finite1 order.strict-implies-order split*: *split-indicator*)
apply(*rule nn-integral-cong*; *auto simp add*: *currentD-finite*[*OF h*] *top-unique less-top*[*symmetric*] *not-less split*: *split-indicator intro*!: *diff-eq-0-ennreal*)
done
also have $\ldots = (\sum^{+} y{\in}range\ Inner.\ f\ n\ (y,\ \langle x\rangle))$ **using** *A*
apply(*simp add*: *nn-integral-count-space-reindex cong*: *nn-integral-cong-simp outgoing-def*)
apply(*auto simp add*: *nn-integral-count-space-indicator outgoing-def A-in max.commute currentD-outside*[*OF h*] *intro*!: *nn-integral-cong split*: *split-indicator dest*: *edge-antiparallel*)
done
also have $\ldots + f\ n\ (SINK,\ \langle x\rangle) = (\sum^{+} y.\ f\ n\ (y,\ \langle x\rangle) * indicator\ (range\ Inner)\ y + f\ n\ (y,\ \langle x\rangle) * indicator\ \{SINK\}\ y)$
by(*simp add*: *nn-integral-add nn-integral-count-space-indicator*)
also have $\ldots = d\text{-}IN\ (f\ n)\ \langle x\rangle$
using *A b disjoint* **unfolding** *d-IN-def*
by(*intro nn-integral-cong*)(*auto split*: *split-indicator simp add*: *notin-range-Inner*)
finally show *?thesis* **using** *A* **by** *simp*
next
case (*B x*)

have *finite1*: $(\sum^{+} y.\ h\ n\ (y,\ x) * indicator\ A\ y) \neq \top$ **for** *A n*
using *currentD-finite-IN*[*OF h*, *of n x*, *unfolded d-IN-def*]
by(*rule neq-top-trans*)(*auto intro*!: *nn-integral-mono split*: *split-indicator*)

have *finite-h*[*simp*]: $h\ n\ (y,\ x) < \top$ **for** *y n*
using *finite1*[*of n* {*y*}] **by** (*simp add*: *less-top*)

let *?h0-gt-hn* = $\{y.\ h\ 0\ (y,\ x) > h\ n\ (y,\ x)\}$
let *?h0-le-hn* = $\{y.\ h\ 0\ (y,\ x) \leq h\ n\ (y,\ x)\}$

have *eq*: *d-IN* (*h 0*) *x* + *f n* (*SOURCE*, $\langle x\rangle$) = *d-IN* (*h n*) *x*
proof(*cases x = b*)
case *True* **with** *currentD-finite-IN*[*OF h*, *of - b*] **show** *?thesis*
by(*simp add*: *add-diff-self-ennreal h0-b*)
next
case *False*
with *B SAT* **have** $x \in SAT\ \Gamma\ (h\ n)$ $x \in SAT\ \Gamma\ (h\ 0)$ **by** *auto*
with *B disjoint* **have** *d-IN* (*h n*) *x* = *d-IN* (*h 0*) *x* **by**(*auto simp add*: *currentD-SAT*[*OF h*])
thus *?thesis* **using** *False* **by**(*simp add*: *currentD-finite-IN*[*OF h*])
qed

have *d-IN* (*f n*) $\langle x\rangle$ = ($\sum^+$ *y*. *f n* (*y*, $\langle x\rangle$) $*$ *indicator* (*range Inner*) *y* + *f n* (*y*, $\langle x\rangle$) $*$ *indicator* {*SOURCE*} *y*)
using *B disjoint* **unfolding** *d-IN-def*
by(*intro nn-integral-cong*)(*auto split*: *split-indicator simp add*: *notin-range-Inner*)
also have ... = ($\sum^+$ *y*∈*range Inner*. *f n* (*y*, $\langle x\rangle$)) + *f n* (*SOURCE*, $\langle x\rangle$) **using** *h0-b*[*of n*]
by(*simp add*: *nn-integral-add nn-integral-count-space-indicator max-def*)
also have ($\sum^+$ *y*∈*range Inner*. *f n* (*y*, $\langle x\rangle$)) = ($\sum^+$ *y*. *h 0* (*y*, *x*) − *h n* (*y*, *x*))
using *B disjoint*
apply(*simp add*: *nn-integral-count-space-reindex cong*: *nn-integral-cong-simp outgoing-def*)
apply(*auto simp add*: *nn-integral-count-space-indicator outgoing-def B-out max.commute currentD-outside*[*OF h*] *intro*!: *nn-integral-cong split*: *split-indicator dest*: *edge-antiparallel*)
done
also have ... = ($\sum^+$ *y*. *h 0* (*y*, *x*) $*$ *indicator ?h0-gt-hn y*) − ($\sum^+$ *y*. *h n* (*y*, *x*) $*$ *indicator ?h0-gt-hn y*)
apply(*subst nn-integral-diff*[*symmetric*])
apply(*simp-all add*: *AE-count-space finite1 order.strict-implies-order split*: *split-indicator*)
apply(*rule nn-integral-cong*; *auto simp add*: *currentD-finite*[*OF h*] *top-unique less-top*[*symmetric*] *not-less split*: *split-indicator intro*!: *diff-eq-0-ennreal*)
done
also have *eq-h-0*: ($\sum^+$ *y*. *h 0* (*y*, *x*) $*$ *indicator ?h0-gt-hn y*) = *d-IN* (*h 0*) *x* − ($\sum^+$ *y*. *h 0* (*y*, *x*) $*$ *indicator ?h0-le-hn y*)
unfolding *d-IN-def*
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space finite1 currentD-finite*[*OF h*] *split*: *split-indicator intro*!: *nn-integral-cong*)
done

also have *eq-h-n*: $(\sum^+ y.\ h\ n\ (y,\ x) * indicator\ ?h0\text{-}gt\text{-}hn\ y) = d\text{-}IN\ (h\ n)\ x - (\sum^+ y.\ h\ n\ (y,\ x) * indicator\ ?h0\text{-}le\text{-}hn\ y)$
unfolding *d-IN-def*
apply(*subst nn-integral-diff*[*symmetric*])
apply(*auto simp add*: *AE-count-space finite1 currentD-finite*[*OF h*] *split*: *split-indicator intro*!: *nn-integral-cong*)
done
also have $d\text{-}IN\ (h\ 0)\ x - (\sum^+ y.\ h\ 0\ (y,\ x) * indicator\ ?h0\text{-}le\text{-}hn\ y) - (d\text{-}IN\ (h\ n)\ x - (\sum^+ y.\ h\ n\ (y,\ x) * indicator\ ?h0\text{-}le\text{-}hn\ y)) + f\ n\ (SOURCE,\ \langle x\rangle) =$
$(\sum^+ y.\ h\ n\ (y,\ x) * indicator\ ?h0\text{-}le\text{-}hn\ y) - (\sum^+ y.\ h\ 0\ (y,\ x) * indicator\ ?h0\text{-}le\text{-}hn\ y)$
apply (*subst diff-add-assoc2-ennreal*)
subgoal by (*auto simp add*: *eq-h-0*[*symmetric*] *eq-h-n*[*symmetric*] *split*: *split-indicator intro*!: *nn-integral-mono*)
apply (*subst diff-add-assoc2-ennreal*)
subgoal by (*auto simp*: *d-IN-def split*: *split-indicator intro*!: *nn-integral-mono*)
apply (*subst diff-diff-commute-ennreal*)
apply (*subst diff-diff-ennreal′*)
subgoal
by (*auto simp*: *d-IN-def split*: *split-indicator intro*!: *nn-integral-mono*) []
subgoal
unfolding *eq-h-n*[*symmetric*]
by (*rule add-increasing2*)
(*auto simp add*: *d-IN-def split*: *split-indicator intro*!: *nn-integral-mono*)
apply (*subst diff-add-assoc2-ennreal*[*symmetric*])
unfolding *eq*
using *currentD-finite-IN*[*OF h*]
apply *simp-all*
done
also have $(\sum^+ y.\ h\ n\ (y,\ x) * indicator\ ?h0\text{-}le\text{-}hn\ y) - (\sum^+ y.\ h\ 0\ (y,\ x) * indicator\ ?h0\text{-}le\text{-}hn\ y) = (\sum^+ y.\ h\ n\ (y,\ x) - h\ 0\ (y,\ x))$
apply(*subst nn-integral-diff*[*symmetric*])
apply(*simp-all add*: *AE-count-space max-def finite1 split*: *split-indicator*)
apply(*rule nn-integral-cong*; *auto simp add*: *not-le split*: *split-indicator*)
by (*metis diff-eq-0-ennreal le-less not-le top-greatest*)
also have $\ldots = (\sum^+ y{\in}range\ Inner.\ f\ n\ (\langle x\rangle,\ y))$ **using** *B disjoint*
apply(*simp add*: *nn-integral-count-space-reindex cong*: *nn-integral-cong-simp outgoing-def*)
apply(*auto simp add*: *B-out currentD-outside*[*OF h*] *max.commute intro*!: *nn-integral-cong split*: *split-indicator dest*: *edge-antiparallel*)
done
also have $\ldots = (\sum^+ y.\ f\ n\ (\langle x\rangle,\ y) * indicator\ (range\ Inner)\ y)$
by(*simp add*: *nn-integral-add nn-integral-count-space-indicator max.left-commute max.commute*)
also have $\ldots = d\text{-}OUT\ (f\ n)\ \langle x\rangle$ **using** *B disjoint*
unfolding *d-OUT-def* **by**(*intro nn-integral-cong*)(*auto split*: *split-indicator simp add*: *notin-range-Inner*)
finally show *?thesis* **using** *B* **by**(*simp*)
qed

qed

have *weight* Γ *b* $-$ *d-IN* (*h 0*) *b* = (*SUP n. value-flow* Ψ (*f n*))
using *OUT-f-SOURCE currentD-finite-IN*[*OF h*, *of 0 b*] *IN*
by (*simp add*: *SUP-diff-ennreal less-top*)
also have (*SUP n. value-flow* Ψ (*f n*)) $\leq \alpha$ **unfolding** α*-def*
apply(*rule SUP-least*)
apply(*rule SUP-upper*)
apply(*simp add*: *f*)
done
also have $\alpha \leq$ *weight* Γ *b* $-$ *d-IN* (*h 0*) *b* **unfolding** α*-def*
proof(*rule SUP-least*; *clarsimp*)
fix *f*
assume *f*: *flow* Ψ *f*
have *d-OUT f SOURCE* = ($\sum^+$ *y*. *f* (*SOURCE*, *y*) $*$ *indicator* {$\langle b \rangle$} *y*) **unfolding** *d-OUT-def*
apply(*rule nn-integral-cong*)
subgoal for *x* **using** *flowD-capacity*[*OF f*, *of* (*SOURCE*, *x*)]
by(*auto split*: *split-indicator*)
done
also have ... = *f* (*SOURCE*, $\langle b \rangle$) **by**(*simp add*: *max-def*)
also have ... $\leq$ *weight* Γ *b* $-$ *d-IN* (*h 0*) *b* **using** *flowD-capacity*[*OF f*, *of* (*SOURCE*, $\langle b \rangle$)] **by** *simp*
finally show *d-OUT f SOURCE* $\leq$ ... **.**
qed
ultimately have α: α = *weight* Γ *b* $-$ *d-IN* (*h 0*) *b* **by**(*rule antisym*[*rotated*])
hence α*-finite*: $\alpha \neq \top$ **by** *simp*

from Ψ.*ex-max-flow*
obtain *g* **where** *g*: *flow* Ψ *g*
and *value-g*: *value-flow* Ψ *g* = α
and *IN-g*: $\bigwedge$*x*. *d-IN g x* $\leq$ *value-flow* Ψ *g* **unfolding** α*-def* **by** *blast*

have *g-le-h0*: *g* ($\langle x \rangle$, $\langle y \rangle$) $\leq$ *h 0* (*x*, *y*) **if** *edge* Γ *x y* **for** *x y*
using *flowD-capacity*[*OF g*, *of* ($\langle x \rangle$, $\langle y \rangle$)] *that* **by** *simp*
note [*simp*] = Ψ.*flowD-finite*[*OF g*]

have *g-SOURCE*: *g* (*SOURCE*, $\langle x \rangle$) = (*if x = b then* α *else 0*) **for** *x*
proof(*cases x = b*)
case *True*
have *g* (*SOURCE*, $\langle x \rangle$) = ($\sum^+$ *y*. *g* (*SOURCE*, *y*) $*$ *indicator* {$\langle x \rangle$} *y*) **by**(*simp add*: *max-def*)
also have ... = *value-flow* Ψ *g* **unfolding** *d-OUT-def* **using** *True*
by(*intro nn-integral-cong*)(*auto split*: *split-indicator intro*: Ψ.*flowD-outside*[*OF g*])
finally show *?thesis* **using** *value-g* **by**(*simp add*: *True*)
qed(*simp add*: Ψ.*flowD-outside*[*OF g*])

let *?g* = λ(*x*, *y*). *g* ($\langle y \rangle$, $\langle x \rangle$)

define h' **where** $h' = h\ 0 \oplus\ ?g$
have h'-*simps*: $h'\ (x,\ y) = (\textit{if edge}\ \Gamma\ x\ y\ \textit{then}\ h\ 0\ (x,\ y) + g\ (\langle y\rangle,\ \langle x\rangle) - g\ (\langle x\rangle,\ \langle y\rangle)\ \textit{else}\ 0)$ **for** $x\ y$
by(*simp add*: h'-*def*)

have *OUT*-h'-*B* [*simp*]: *d-OUT* $h'\ x = 0$ **if** $x \in B\ \Gamma$ **for** x **using** *that* **unfolding** *d-OUT-def*
by(*simp add*: *nn-integral-0-iff emeasure-count-space-eq-0*)(*simp add*: h'-*simps B-out*)
have *IN*-h'-*A* [*simp*]: *d-IN* $h'\ x = 0$ **if** $x \in A\ \Gamma$ **for** x **using** *that* **unfolding** *d-IN-def*
by(*simp add*: *nn-integral-0-iff emeasure-count-space-eq-0*)(*simp add*: h'-*simps A-in*)
have h'-*outside*: $h'\ e = 0$ **if** $e \notin$ **E** **for** e **unfolding** h'-*def* **using** *that* **by**(*rule plus-flow-outside*)
have *OUT*-h'-*outside*: *d-OUT* $h'\ x = 0$ **and** *IN*-h'-*outside*: *d-IN* $h'\ x = 0$ **if** $x \notin$ **V** **for** x **using** *that*
by(*auto simp add*: *d-OUT-def d-IN-def nn-integral-0-iff emeasure-count-space-eq-0 vertex-def intro*: h'-*outside*)

have *g-le-OUT*: $g\ (\textit{SINK},\ \langle x\rangle) \le$ *d-OUT* $g\ \langle x\rangle$ **for** x
by (*subst flowD-KIR*[*OF g*]) (*simp-all add*: *d-IN-ge-point*)

have *OUT-g-A*: *d-OUT* $?g\ x =$ *d-OUT* $g\ \langle x\rangle - g\ (\textit{SINK},\ \langle x\rangle)$ **if** $x \in A\ \Gamma$ **for** x
proof −
have *d-OUT* $?g\ x = (\sum^+ y{\in}\textit{range Inner}.\ g\ (y,\ \langle x\rangle))$
by(*simp add*: *nn-integral-count-space-reindex d-OUT-def*)
also have … = *d-IN* $g\ \langle x\rangle - (\sum^+ y.\ g\ (y,\ \langle x\rangle) * \textit{indicator}\ \{\textit{SINK}\}\ y)$ **unfolding** *d-IN-def*
using *that b disjoint flowD-capacity*[*OF g*, *of* (*SOURCE*, $\langle x\rangle$)]
by(*subst nn-integral-diff*[*symmetric*])
(*auto simp add*: *nn-integral-count-space-indicator notin-range-Inner max-def intro*!: *nn-integral-cong split*: *split-indicator if-split-asm*)
also have … = *d-OUT* $g\ \langle x\rangle - g\ (\textit{SINK},\ \langle x\rangle)$ **by**(*simp add*: *flowD-KIR*[*OF g*] *max-def*)
finally show *?thesis* **.**
qed
have *IN-g-A*: *d-IN* $?g\ x =$ *d-OUT* $g\ \langle x\rangle - g\ (\langle x\rangle,\ \textit{SINK})$ **if** $x \in A\ \Gamma$ **for** x
proof −
have *d-IN* $?g\ x = (\sum^+ y{\in}\textit{range Inner}.\ g\ (\langle x\rangle,\ y))$
by(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
also have … = *d-OUT* $g\ \langle x\rangle - (\sum^+ y.\ g\ (\langle x\rangle,\ y) * \textit{indicator}\ \{\textit{SINK}\}\ y)$ **unfolding** *d-OUT-def*
using *that b disjoint flowD-capacity*[*OF g*, *of* ($\langle x\rangle$, *SOURCE*)]
by(*subst nn-integral-diff*[*symmetric*])
(*auto simp add*: *nn-integral-count-space-indicator notin-range-Inner max-def intro*!: *nn-integral-cong split*: *split-indicator if-split-asm*)
also have … = *d-OUT* $g\ \langle x\rangle - g\ (\langle x\rangle,\ \textit{SINK})$ **by**(*simp add*: *max-def*)

finally show *?thesis* **.**
qed
have *OUT-g-B*: *d-OUT ?g x* = *d-IN g* ⟨*x*⟩ − *g* (*SOURCE*, ⟨*x*⟩) **if** *x* ∈ *B* Γ **for** *x*
proof −
have *d-OUT ?g x* = ($\sum^+$ *y*∈*range Inner*. *g* (*y*, ⟨*x*⟩))
by(*simp add*: *nn-integral-count-space-reindex d-OUT-def*)
also have ... = *d-IN g* ⟨*x*⟩ − ($\sum^+$ *y*. *g* (*y*, ⟨*x*⟩) ∗ *indicator* {*SOURCE*} *y*)
unfolding *d-IN-def*
using *that b disjoint flowD-capacity*[*OF g*, *of* (*SINK*, ⟨*x*⟩)]
by(*subst nn-integral-diff*[*symmetric*])
(*auto simp add*: *nn-integral-count-space-indicator notin-range-Inner max-def intro*!: *nn-integral-cong split*: *split-indicator if-split-asm*)
also have ... = *d-IN g* ⟨*x*⟩ − *g* (*SOURCE*, ⟨*x*⟩) **by**(*simp add*: *max-def*)
finally show *?thesis* **.**
qed
have *IN-g-B*: *d-IN ?g x* = *d-OUT g* ⟨*x*⟩ **if** *x* ∈ *B* Γ **for** *x*
proof −
have *d-IN ?g x* = ($\sum^+$ *y*∈*range Inner*. *g* (⟨*x*⟩, *y*))
by(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
also have ... = *d-OUT g* ⟨*x*⟩ **unfolding** *d-OUT-def* **using** *that disjoint*
by(*auto 4 3 simp add*: *nn-integral-count-space-indicator notin-range-Inner intro*!: *nn-integral-cong* Ψ.*flowD-outside*[*OF g*] *split*: *split-indicator*)
finally show *?thesis* **.**
qed

have *finite-g-IN*: *d-IN ?g x* ≠ ⊤ **for** *x* **using** *α-finite*
proof(*rule neq-top-trans*)
have *d-IN ?g x* = ($\sum^+$ *y*∈*range Inner*. *g* (⟨*x*⟩, *y*))
by(*auto simp add*: *d-IN-def nn-integral-count-space-reindex*)
also have ... ≤ *d-OUT g* ⟨*x*⟩ **unfolding** *d-OUT-def*
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... = *d-IN g* ⟨*x*⟩ **by**(*rule flowD-KIR*[*OF g*]) *simp-all*
also have ... ≤ α **using** *IN-g value-g* **by** *simp*
finally show *d-IN ?g x* ≤ α **.**
qed

have *OUT-h′-A*: *d-OUT h′ x* = *d-OUT* (*h 0*) *x* + *g* (⟨*x*⟩, *SINK*) − *g* (*SINK*, ⟨*x*⟩) **if** *x* ∈ *A* Γ **for** *x*
proof −
have *d-OUT h′ x* = *d-OUT* (*h 0*) *x* + ($\sum^+$ *y*. *?g* (*x*, *y*) ∗ *indicator* **E** (*x*, *y*)) − ($\sum^+$ *y*. *?g* (*y*, *x*) ∗ *indicator* **E** (*x*, *y*))
unfolding *h′-def*
apply(*subst OUT-plus-flow*[*of* Γ *h 0 ?g*, *OF currentD-outside′*[*OF h*]])
apply(*auto simp add*: *g-le-h0 finite-g-IN*)
done
also have ($\sum^+$ *y*. *?g* (*x*, *y*) ∗ *indicator* **E** (*x*, *y*)) = *d-OUT ?g x* **unfolding** *d-OUT-def* **using** *that*
by(*auto simp add*: *A-in split*: *split-indicator intro*!: *nn-integral-cong* Ψ.*flowD-outside*[*OF*

g])
also have ... $=$ *d-OUT* g $\langle x\rangle$ $-$ g (*SINK*, $\langle x\rangle$) **using** *that* **by**(*rule OUT-g-A*)
also have ($\sum^+$ y. *?g* (y, x) $*$ *indicator* **E** (x, y)) $=$ *d-IN ?g* x **using** *that* **unfolding** *d-IN-def*
by(*auto simp add*: *A-in split*: *split-indicator intro*!: *nn-integral-cong* Ψ.*flowD-outside*[*OF* g])
also have ... $=$ *d-OUT* g $\langle x\rangle$ $-$ g ($\langle x\rangle$, *SINK*) **using** *that* **by**(*rule IN-g-A*)
also have *d-OUT* (h *0*) x $+$ (*d-OUT* g $\langle x\rangle$ $-$ g (*SINK*, $\langle x\rangle$)) $-$ ... $=$ *d-OUT* (h *0*) x $+$ g ($\langle x\rangle$, *SINK*) $-$ g (*SINK*, $\langle x\rangle$)
apply(*simp add*: *g-le-OUT add-diff-eq-ennreal d-OUT-ge-point*)
apply(*subst diff-diff-commute-ennreal*)
apply(*simp add*: *add-increasing d-OUT-ge-point g-le-OUT diff-diff-ennreal'*)
apply(*subst add.assoc*)
apply(*subst* (*2*) *add.commute*)
apply(*subst add.assoc*[*symmetric*])
apply(*subst ennreal-add-diff-cancel-right*)
apply(*simp-all add*: Ψ.*flowD-finite-OUT*[*OF* g])
done
finally show *?thesis* **.**
qed

have *finite-g-OUT*: *d-OUT ?g* $x \neq \top$ **for** x **using** α*-finite*
proof(*rule neq-top-trans*)
have *d-OUT ?g* x $=$ ($\sum^+$ $y{\in}$*range Inner*. g (y, $\langle x\rangle$))
by(*auto simp add*: *d-OUT-def nn-integral-count-space-reindex*)
also have ... $\leq$ *d-IN* g $\langle x\rangle$ **unfolding** *d-IN-def*
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... $\leq \alpha$ **using** *IN-g value-g* **by** *simp*
finally show *d-OUT ?g* $x \leq \alpha$ **.**
qed

have *IN-h'-B*: *d-IN* h' x $=$ *d-IN* (h *0*) x $+$ g (*SOURCE*, $\langle x\rangle$) **if** $x \in B\ \Gamma$ **for** x
proof $-$
have *g-le*: g (*SOURCE*, $\langle x\rangle$) $\leq$ *d-IN* g $\langle x\rangle$
by (*rule d-IN-ge-point*)

have *d-IN* h' x $=$ *d-IN* (h *0*) x $+$ ($\sum^+$ y. g ($\langle x\rangle$, $\langle y\rangle$) $*$ *indicator* **E** (y, x)) $-$ ($\sum^+$ y. g ($\langle y\rangle$, $\langle x\rangle$) $*$ *indicator* **E** (y, x))
unfolding *h'-def*
by(*subst IN-plus-flow*[*of* Γ h *0 ?g*, *OF currentD-outside'*[*OF* h]])
(*auto simp add*: *g-le-h0 finite-g-OUT*)
also have ($\sum^+$ y. g ($\langle x\rangle$, $\langle y\rangle$) $*$ *indicator* **E** (y, x)) $=$ *d-IN ?g* x **unfolding** *d-IN-def* **using** *that*
by(*intro nn-integral-cong*)(*auto split*: *split-indicator intro*!: Ψ.*flowD-outside*[*OF* g] *simp add*: *B-out*)
also have ... $=$ *d-OUT* g $\langle x\rangle$ **using** *that* **by**(*rule IN-g-B*)
also have ($\sum^+$ y. g ($\langle y\rangle$, $\langle x\rangle$) $*$ *indicator* **E** (y, x)) $=$ *d-OUT ?g* x **unfolding** *d-OUT-def* **using** *that*

by(*intro nn-integral-cong*)(*auto split*: *split-indicator intro*!: Ψ.*flowD-outside*[*OF g*] *simp add*: *B-out*)
also have ... = *d-IN g* ⟨*x*⟩ − *g* (*SOURCE*, ⟨*x*⟩) **using** *that* **by**(*rule OUT-g-B*)
also have *d-IN* (*h 0*) *x* + *d-OUT g* ⟨*x*⟩ − ... = *d-IN* (*h 0*) *x* + *g* (*SOURCE*, ⟨*x*⟩)
using Ψ.*flowD-finite-IN*[*OF g*] *g-le*
by(*cases d-IN* (*h 0*) *x*; *cases d-IN g* ⟨*x*⟩; *cases d-IN g* ⟨*x*⟩; *cases g* (*SOURCE*, ⟨*x*⟩))
(*auto simp*: *flowD-KIR*[*OF g*] *top-add ennreal-minus-if ennreal-plus-if simp del*: *ennreal-plus*)
finally show *?thesis* **.**
qed

have *h'*: *current* Γ *h'*
proof
fix *x*
consider (*A*) *x* ∈ *A* Γ | (*B*) *x* ∈ *B* Γ | (*outside*) *x* ∉ **V** **using** *bipartite-V* **by** *auto*
note *cases* = *this*

show *d-OUT h' x* ≤ *weight* Γ *x*
proof(*cases rule*: *cases*)
case *A*
then have *d-OUT h' x* = *d-OUT* (*h 0*) *x* + *g* (⟨*x*⟩, *SINK*) − *g* (*SINK*, ⟨*x*⟩) **by**(*simp add*: *OUT-h'-A*)
also have ... ≤ *d-OUT* (*h 0*) *x* + *g* (⟨*x*⟩, *SINK*) **by**(*rule diff-le-self-ennreal*)
also have *g* (⟨*x*⟩, *SINK*) ≤ *weight* Γ *x* − *d-OUT* (*h 0*) *x*
using *flowD-capacity*[*OF g*, *of* (⟨*x*⟩, *SINK*)] *A* **by** *simp*
also have *d-OUT* (*h 0*) *x* + ... = *weight* Γ *x*
by(*simp add*: *add-diff-eq-ennreal add-diff-inverse-ennreal currentD-finite-OUT*[*OF h*] *currentD-weight-OUT*[*OF h*])
finally show *?thesis* **by**(*simp add*: *add-left-mono*)
qed(*simp-all add*: *OUT-h'-outside*)

show *d-IN h' x* ≤ *weight* Γ *x*
proof(*cases rule*: *cases*)
case *B*
hence *d-IN h' x* = *d-IN* (*h 0*) *x* + *g* (*SOURCE*, ⟨*x*⟩) **by**(*rule IN-h'-B*)
also have ... ≤ *weight* Γ *x*
by(*simp add*: *g-SOURCE* α *currentD-weight-IN*[*OF h*] *add-diff-eq-ennreal add-diff-inverse-ennreal currentD-finite-IN*[*OF h*])
finally show *?thesis* **.**
qed(*simp-all add*: *IN-h'-outside*)
next
show *h' e* = *0* **if** *e* ∉ **E** **for** *e* **using** *that* **by**(*simp split*: *prod.split-asm add*: *h'-simps*)
qed
moreover
have *SAT-h'*: *B* Γ ∩ **V** ⊆ *SAT* Γ *h'*

proof
show $x \in SAT\ \Gamma\ h'$ **if** $x \in B\ \Gamma \cap \mathbf{V}$ **for** x **using** *that*
proof(*cases* $x = b$)
case *True*
have $d\text{-}IN\ h'\ x = weight\ \Gamma\ x$ **using** *that True*
by(*simp add*: *IN-h′-B g-SOURCE α currentD-weight-IN*[*OF h*] *add-diff-eq-ennreal add-diff-inverse-ennreal currentD-finite-IN*[*OF h*])
thus *?thesis* **by**(*simp add*: *SAT.simps*)
next
case *False*
have $d\text{-}IN\ h'\ x = d\text{-}IN\ (h\ 0)\ x$ **using** *that False* **by**(*simp add*: *IN-h′-B g-SOURCE*)
also have $\ldots = weight\ \Gamma\ x$
using *SAT*[*of 0*, *THEN subsetD*, *of x*] *False that currentD-SAT*[*OF h*, *of x 0*] *disjoint* **by** *auto*
finally show *?thesis* **by**(*simp add*: *SAT.simps*)
qed
qed
moreover
have *wave* $\Gamma\ h'$
proof
have *separating* $\Gamma\ (B\ \Gamma \cap \mathbf{V})$
proof
fix *x y p*
assume *x*: $x \in A\ \Gamma$ **and** *y*: $y \in B\ \Gamma$ **and** *p*: *path* $\Gamma\ x\ p\ y$
hence *Nil*: $p \neq []$ **using** *disjoint* **by**(*auto simp add*: *rtrancl-path-simps*)
from *rtrancl-path-last*[*OF p Nil*] *last-in-set*[*OF Nil*] *y rtrancl-path-Range*[*OF p*, *of y*]
show $(\exists z \in set\ p.\ z \in B\ \Gamma \cap \mathbf{V}) \lor x \in B\ \Gamma \cap \mathbf{V}$ **by**(*auto intro*: *vertexI2*)
qed
moreover have *TER*: $B\ \Gamma \cap \mathbf{V} \subseteq TER\ h'$ **using** *SAT-h′* **by**(*auto simp add*: *SINK*)
ultimately show *separating* Γ (*TER* h') **by**(*rule separating-weakening*)
qed(*rule* h')
ultimately show *?thesis* **by** *blast*
qed

end

lemma *countable-bipartite-web-reduce-weight*:
assumes $weight\ \Gamma\ x \geq w$
shows *countable-bipartite-web* (*reduce-weight* $\Gamma\ x\ w$)
using *bipartite-V A-vertex bipartite-E disjoint assms*
by *unfold-locales* (*auto 4 3 simp add*: *weight-outside*)

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* Γ *b* ε)
proof(*rule ccontr*)
assume $\neg$ *?thesis*
hence *hindered*: *hindered* (*reduce-weight* Γ *b* ε) **if** $\varepsilon > 0$ $\varepsilon < \delta$ **for** ε **using** *that* **by** *simp*

from *b disjoint* **have** *bnA*: $b \notin A\ \Gamma$ **by** *blast*

define *wb* **where** *wb* = *enn2real* (*weight* Γ *b*)
have *wb-conv*: *weight* Γ *b* = *ennreal wb* **by**(*simp add*: *wb-def less-top*[*symmetric*])
have *wb-pos*: *wb* > 0 **using** *wb* **by**(*simp add*: *wb-conv*)

define ε **where** $\varepsilon\ n = min\ \delta\ wb\ /\ (n + 2)$ **for** $n :: nat$
have ε*-pos*: $\varepsilon\ n > 0$ **for** *n* **using** *wb-pos* δ **by**(*simp add*: ε*-def*)
have ε*-nonneg*: $0 \le \varepsilon\ n$ **for** *n* **using** ε*-pos*[*of n*] **by** *simp*
have $*$: $\varepsilon\ n \le min\ wb\ \delta\ /\ 2$ **for** *n* **using** *wb-pos* δ
by(*auto simp add*: ε*-def field-simps min-def*)
have ε*-le*: $\varepsilon\ n \le wb$ **and** ε*-less*: $\varepsilon\ n < wb$ **and** ε*-less-*δ: $\varepsilon\ n < \delta$ **and** ε*-le*$'$: $\varepsilon\ n \le wb\ /\ 2$ **for** *n*
using $*$[*of n*] ε*-pos*[*of n*] **by**(*auto*)

define Γ' **where** $\Gamma'\ n$ = *reduce-weight* Γ *b* (ε *n*) **for** $n :: nat$
have Γ'*-sel* [*simp*]:
edge (Γ' *n*) = *edge* Γ
A (Γ' *n*) = *A* Γ
B (Γ' *n*) = *B* Γ
weight (Γ' *n*) *x* = *weight* Γ *x* $-$ (*if x* = *b then* ε *n else* 0)
essential (Γ' *n*) = *essential* Γ
roofed-gen (Γ' *n*) = *roofed-gen* Γ
for *n x* **by**(*simp-all add*: Γ'*-def*)

have *vertex-*Γ' [*simp*]: *vertex* (Γ' *n*) = *vertex* Γ **for** *n*
by(*simp add*: *vertex-def fun-eq-iff*)

from *wb* **have** $b \in$ **V** **using** *weight-outside*[*of b*] **by**(*auto intro*: *ccontr*)
interpret Γ': *countable-bipartite-web* Γ' *n* **for** *n* **unfolding** Γ'*-def*
using *wb-pos* **by**(*intro countable-bipartite-web-reduce-weight*)(*simp-all add*: *wb-conv* ε*-le* ε*-nonneg*)

obtain *g* **where** *g*: $\bigwedge n.$ *current* (Γ' *n*) (*g n*)
and *w*: $\bigwedge n.$ *wave* (Γ' *n*) (*g n*)
and *hind*: $\bigwedge n.$ *hindrance* (Γ' *n*) (*g n*) **using** *hindered*[*OF* ε*-pos*, *unfolded wb-conv ennreal-less-iff*, *OF* ε*-less-*δ]
unfolding *hindered.simps* Γ'*-def* **by** *atomize-elim metis*
from *g* **have** *g*Γ: *current* Γ (*g n*) **for** *n*
by(*rule current-weight-mono*)(*auto simp add*: ε*-nonneg diff-le-self-ennreal*)
note [*simp*] = *currentD-finite*[*OF g*Γ]

have *b-TER*: $b \in TER_{\Gamma' \, n}$ $(g\ n)$ **for** n
proof(*rule ccontr*)
assume b': $b \notin TER_{\Gamma' \, n}$ $(g\ n)$
then have *TER*: $TER_{\Gamma' \, n}$ $(g\ n)$ = *TER* $(g\ n)$ **using** *b* *ε-nonneg*[*of n*]
by(*auto simp add*: *SAT.simps split*: *if-split-asm intro*: *ennreal-diff-le-mono-left*)
from *w*[*of n*] *TER* **have** *wave* Γ (*g n*) **by**(*simp add*: *wave.simps separating-gen.simps*)
moreover have *hindrance* Γ (*g n*) **using** *hind*[*of n*] *TER bnA b′*
by(*auto simp add*: *hindrance.simps split*: *if-split-asm*)
ultimately show *False* **using** *loose-unhindered*[*OF loose*] *g*Γ[*of n*] **by**(*auto intro*: *hindered.intros*)
qed

have *IN-g-b*: *d-IN* (*g n*) *b* = *weight* Γ *b* − ε *n* **for** *n* **using** *b-TER*[*of n*] *bnA*
by(*auto simp add*: *currentD-SAT*[*OF g*])

define *factor* **where** *factor n* = (*wb* − ε *0*) / (*wb* − ε *n*) **for** *n*
have *factor-le-1*: *factor n* ≤ *1* **for** *n* **using** *wb-pos* δ *ε-less*[*of n*]
by(*auto simp add*: *factor-def field-simps ε-def min-def*)
have *factor-pos*: *0* < *factor n* **for** *n* **using** *wb-pos* δ ∗ *ε-less* **by**(*simp add*: *factor-def field-simps*)
have *factor*: (*wb* − ε *n*) ∗ *factor n* = *wb* − ε *0* **for** *n* **using** *ε-less*[*of n*]
by(*simp add*: *factor-def field-simps*)

define *g′* **where** *g′* = (λ*n* (*x*, *y*). *if y* = *b then g n* (*x*, *y*) ∗ *factor n else g n* (*x*, *y*))
have *g′-simps*: *g′ n* (*x*, *y*) = (*if y* = *b then g n* (*x*, *y*) ∗ *factor n else g n* (*x*, *y*)) **for** *n x y* **by**(*simp add*: *g′-def*)
have *g′-le-g*: *g′ n e* ≤ *g n e* **for** *n e* **using** *factor-le-1*[*of n*]
by(*cases e g n e rule*: *prod.exhaust*[*case-product ennreal-cases*])
(*auto simp add*: *g′-simps field-simps mult-left-le*)

have *4* + (*n* ∗ *6* + *n* ∗ (*n* ∗ *2*)) ≠ (*0* :: *real*) **for** *n* :: *nat*
by(*metis* (*mono-tags*, *opaque-lifting*) *add-is-0 of-nat-eq-0-iff of-nat-numeral zero-neq-numeral*)
then have *IN-g′*: *d-IN* (*g′ n*) *x* = (*if x* = *b then weight* Γ *b* − ε *0 else d-IN* (*g n*) *x*) **for** *x n*
using *b-TER*[*of n*] *bnA factor-pos*[*of n*] *factor*[*of n*] *wb-pos* δ
by(*auto simp add*: *d-IN-def g′-simps nn-integral-divide nn-integral-cmult currentD-SAT*[*OF g*] *wb-conv ε-def field-simps*
ennreal-minus ennreal-mult′[*symmetric*] *intro*!: *arg-cong*[**where** *f*=*ennreal*])
have *OUT-g′*: *d-OUT* (*g′ n*) *x* = *d-OUT* (*g n*) *x* − *g n* (*x*, *b*) ∗ (*1* − *factor n*) **for** *n x*
proof −
have *d-OUT* (*g′ n*) *x* = ($\sum^+$ *y*. *g n* (*x*, *y*)) − ($\sum^+$ *y*. (*g n* (*x*, *y*) ∗ (*1* − *factor n*)) ∗ *indicator* {*b*} *y*)
using *factor-le-1*[*of n*] *factor-pos*[*of n*]
apply(*cases g n* (*x*, *b*))

apply(*subst nn-integral-diff* [*symmetric*])
apply(*auto simp add*: g'*-simps nn-integral-divide d-OUT-def AE-count-space mult-left-le ennreal-mult-eq-top-iff*
ennreal-mult'[*symmetric*] *ennreal-minus-if*
intro!: *nn-integral-cong split*: *split-indicator*)
apply(*simp-all add*: *field-simps*)
done
also have $\ldots = \textit{d-OUT}\ (g\ n)\ x - g\ n\ (x,\ b) * (1 - \textit{factor}\ n)$ **using** *factor-le-1* [*of n*]
by(*subst nn-integral-indicator-singleton*)(*simp-all add*: *d-OUT-def field-simps*)
finally show *?thesis* **.**
qed

have g': *current* $(\Gamma'\ 0)\ (g'\ n)$ **for** n
proof
show $\textit{d-OUT}\ (g'\ n)\ x \leq \textit{weight}\ (\Gamma'\ 0)\ x$ **for** x
using *b-TER*[*of n*] *currentD-weight-OUT*[*OF g*, *of n x*] ε*-le*[*of 0*] *factor-le-1* [*of n*]
by(*auto simp add*: *OUT-g' SINK.simps ennreal-diff-le-mono-left*)
show $\textit{d-IN}\ (g'\ n)\ x \leq \textit{weight}\ (\Gamma'\ 0)\ x$ **for** x
using *d-IN-mono*[*of g' n x*, *OF g'-le-g*] *currentD-weight-IN*[*OF g*, *of n x*] *b-TER*[*of n*] *b*
by(*auto simp add*: *IN-g' SAT.simps wb-conv* ε*-def*)
show $g'\ n\ e = 0$ **if** $e \notin \mathbf{E}_{\Gamma'\ 0}$ **for** e **using** *that* **by**(*cases e*)(*clarsimp simp add*: g'*-simps currentD-outside*[*OF g*])
qed

have *SINK-g'*: $\textit{SINK}\ (g\ n) = \textit{SINK}\ (g'\ n)$ **for** n **using** *factor-pos*[*of n*]
by(*auto simp add*: *SINK.simps currentD-OUT-eq-0*[*OF g*] *currentD-OUT-eq-0*[*OF g'*] g'*-simps split*: *if-split-asm*)
have *SAT-g'*: $\textit{SAT}\ (\Gamma'\ n)\ (g\ n) = \textit{SAT}\ (\Gamma'\ 0)\ (g'\ n)$ **for** n **using** *b-TER*[*of n*] ε*-le'*[*of 0*]
by(*auto simp add*: *SAT.simps wb-conv IN-g' IN-g-b*)
have *TER-g'*: $\textit{TER}_{\Gamma'\ n}\ (g\ n) = \textit{TER}_{\Gamma'\ 0}\ (g'\ n)$ **for** n
using *b-TER*[*of n*] **by**(*auto simp add*: *SAT.simps SINK-g' OUT-g' IN-g' wb-conv* ε*-def*)

have w': *wave* $(\Gamma'\ 0)\ (g'\ n)$ **for** n
proof
have *separating* $(\Gamma'\ 0)\ (\textit{TER}_{\Gamma'\ n}\ (g\ n))$ **using** *waveD-separating*[*OF w*, *of n*]
by(*simp add*: *separating-gen.simps*)
then show *separating* $(\Gamma'\ 0)\ (\textit{TER}_{\Gamma'\ 0}\ (g'\ n))$ **unfolding** *TER-g'* **.**
qed(*rule g'*)

define f **where** $f = \textit{rec-nat}\ (g\ 0)\ (\lambda n\ \textit{rec}.\ \textit{rec} \frown_{\Gamma'\ 0} g'\ (n + 1))$
have *f-simps* [*simp*]:
$f\ 0 = g\ 0$
$f\ (\textit{Suc}\ n) = f\ n \frown_{\Gamma'\ 0} g'\ (n + 1)$
for n **by**(*simp-all add*: *f-def*)

have *f*: *current* $(\Gamma'\ \mathit{0})$ $(f\ n)$ **and** *fw*: *wave* $(\Gamma'\ \mathit{0})$ $(f\ n)$ **for** *n*
proof(*induction n*)
case (*Suc n*)
{ **case** *1* **show** *?case* **unfolding** *f-simps* **using** *Suc.IH* g' **by**(*rule current-plus-web*)
}
{ **case** *2* **show** *?case* **unfolding** *f-simps* **using** *Suc.IH* g' w' **by**(*rule wave-plus'*)
}
qed(*simp-all add*: *g w*)

have *f-inc*: $n \le m \Longrightarrow f\ n \le f\ m$ **for** *n m*
proof(*induction m rule*: *dec-induct*)
case (*step k*)
note *step.IH*
also have $f\ k \le (f\ k \frown_{\Gamma'\ \mathit{0}} g'\ (k + \mathit{1}))$
by(*rule le-funI plus-web-greater*)+
also have $\ldots = f\ (Suc\ k)$ **by** *simp*
finally show *?case* **.**
qed *simp*
have *chain-f*: *Complete-Partial-Order.chain* $(\le)$ (*range f*)
by(*rule chain-imageI*[**where** *le-a*=$(\le)$])(*simp-all add*: *f-inc*)
have *countable* (*support-flow* (*f n*)) **for** *n* **using** *current-support-flow*[*OF f*, *of n*]
by(*rule countable-subset*) *simp*
hence *supp-f*: *countable* (*support-flow* (*SUP n. f n*)) **by**(*subst support-flow-Sup*)*simp*

have *RF-f*: $RF\ (TER_{\Gamma'\ \mathit{0}}\ (f\ n)) = RF\ (\bigcup i{\le}n.\ TER_{\Gamma'\ \mathit{0}}\ (g'\ i))$ **for** *n*
proof(*induction n*)
case *0* **show** *?case* **by**(*simp add*: *TER-g'*)
next
case (*Suc n*)
have $RF\ (TER_{\Gamma'\ \mathit{0}}\ (f\ (Suc\ n))) = RF_{\Gamma'\ \mathit{0}}\ (TER_{\Gamma'\ \mathit{0}}\ (f\ n \frown_{\Gamma'\ \mathit{0}} g'\ (n + \mathit{1})))$
by *simp*
also have $\ldots = RF_{\Gamma'\ \mathit{0}}\ (TER_{\Gamma'\ \mathit{0}}\ (f\ n) \cup TER_{\Gamma'\ \mathit{0}}\ (g'\ (n + \mathit{1})))$ **using** *f fw*
g' w'
by(*rule RF-TER-plus-web*)
also have $\ldots = RF_{\Gamma'\ \mathit{0}}\ (RF_{\Gamma'\ \mathit{0}}\ (TER_{\Gamma'\ \mathit{0}}\ (f\ n)) \cup TER_{\Gamma'\ \mathit{0}}\ (g'\ (n + \mathit{1})))$
by(*simp add*: *roofed-idem-Un1*)
also have $RF_{\Gamma'\ \mathit{0}}\ (TER_{\Gamma'\ \mathit{0}}\ (f\ n)) = RF_{\Gamma'\ \mathit{0}}\ (\bigcup i{\le}n.\ TER_{\Gamma'\ \mathit{0}}\ (g'\ i))$ **by**(*simp*
add: *Suc.IH*)
also have $RF_{\Gamma'\ \mathit{0}}\ (\ldots \cup TER_{\Gamma'\ \mathit{0}}\ (g'\ (n + \mathit{1}))) = RF_{\Gamma'\ \mathit{0}}\ ((\bigcup i{\le}n.\ TER_{\Gamma'\ \mathit{0}}$
$(g'\ i)) \cup TER_{\Gamma'\ \mathit{0}}\ (g'\ (n + \mathit{1})))$
by(*simp add*: *roofed-idem-Un1*)
also have $(\bigcup i{\le}n.\ TER_{\Gamma'\ \mathit{0}}\ (g'\ i)) \cup TER_{\Gamma'\ \mathit{0}}\ (g'\ (n + \mathit{1})) = (\bigcup i{\le}Suc\ n.$
$TER_{\Gamma'\ \mathit{0}}\ (g'\ i))$
unfolding *atMost-Suc UN-insert* **by**(*simp add*: *Un-commute*)
finally show *?case* **by** *simp*
qed

define $g\omega$ **where** $g\omega = (SUP\ n.\ f\ n)$

have $g\omega$: *current* $(\Gamma'\ 0)\ g\omega$ **unfolding** $g\omega$*-def* **using** *chain-f*
by(*rule current-Sup*)(*auto simp add*: *f supp-f*)
have $w\omega$: *wave* $(\Gamma'\ 0)\ g\omega$ **unfolding** $g\omega$*-def* **using** *chain-f*
by(*rule wave-lub*)(*auto simp add*: *fw supp-f*)
from $g\omega$ **have** $g\omega'$: *current* $(\Gamma'\ n)\ g\omega$ **for** n **using** *wb-pos* δ
by(*elim current-weight-mono*)(*auto simp add*: ε*-le wb-conv* ε*-def field-simps ennreal-minus-if min-le-iff-disj*)

have *SINK-*$g\omega$: *SINK* $g\omega = (\bigcap n.\ \mathit{SINK}\ (f\ n))$ **unfolding** $g\omega$*-def*
by(*subst SINK-Sup*[*OF chain-f*])(*simp-all add*: *supp-f*)
have *SAT-*$g\omega$: *SAT* $(\Gamma'\ 0)\ (f\ n) \subseteq$ *SAT* $(\Gamma'\ 0)\ g\omega$ **for** n
unfolding $g\omega$*-def* **by**(*rule SAT-Sup-upper*) *simp*

have *g-b-out*: $g\ n\ (b,\ x) = 0$ **for** $n\ x$ **using** *b-TER*[*of n*] **by**(*simp add*: *SINK.simps currentD-OUT-eq-0*[*OF g*])
have g'*-b-out*: $g'\ n\ (b,\ x) = 0$ **for** $n\ x$ **by**(*simp add*: g'*-simps g-b-out*)
have $f\ n\ (b,\ x) = 0$ **for** $n\ x$ **by**(*induction n*)(*simp-all add*: *g-b-out* g'*-b-out*)
hence *b-SINK-f*: $b \in$ *SINK* $(f\ n)$ **for** n **by**(*simp add*: *SINK.simps d-OUT-def*)
hence *b-SINK-*$g\omega$: $b \in$ *SINK* $g\omega$ **by**(*simp add*: *SINK-*$g\omega$)

have *RF-circ*: $RF^{\circ}{}_{\Gamma'\ n}\ (TER_{\Gamma'\ 0}\ (g'\ n)) = RF^{\circ}{}_{\Gamma'\ 0}\ (TER_{\Gamma'\ 0}\ (g'\ n))$ **for** n **by**(*simp add*: *roofed-circ-def*)
have *edge-restrict-*Γ': *edge* (*quotient-web* $(\Gamma'\ 0)\ (g'\ n)$) = *edge* (*quotient-web* $(\Gamma'\ n)\ (g\ n)$) **for** n
by(*simp add*: *fun-eq-iff TER-*g' *RF-circ*)
have *restrict-curr-*g': $f \upharpoonleft \Gamma'\ 0\ /\ g'\ n = f \upharpoonleft \Gamma'\ n\ /\ g\ n$ **for** $n\ f$
by(*simp add*: *restrict-current-def RF-circ TER-*g')

have *RF-restrict*: *roofed-gen* (*quotient-web* $(\Gamma'\ n)\ (g\ n)$) = *roofed-gen* (*quotient-web* $(\Gamma'\ 0)\ (g'\ n)$) **for** n
by(*simp add*: *roofed-def fun-eq-iff edge-restrict-*Γ')

have $g\omega r'$: *current* (*quotient-web* $(\Gamma'\ 0)\ (g'\ n)$) $(g\omega \upharpoonleft \Gamma'\ 0\ /\ g'\ n)$ **for** n **using** $w'\ g\omega$
by(*rule current-restrict-current*)
have $g\omega r$: *current* (*quotient-web* $(\Gamma'\ n)\ (g\ n)$) $(g\omega \upharpoonleft \Gamma'\ n\ /\ g\ n)$ **for** n **using** $w\ g\omega'$
by(*rule current-restrict-current*)
have $w\omega r$: *wave* (*quotient-web* $(\Gamma'\ n)\ (g\ n)$) $(g\omega \upharpoonleft \Gamma'\ n\ /\ g\ n)$ (**is** *wave* $?\Gamma'\ ?g\omega$) **for** n
proof
have $*$: *wave* (*quotient-web* $(\Gamma'\ 0)\ (g'\ n)$) $(g\omega \upharpoonleft \Gamma'\ 0\ /\ g'\ n)$
using $g'\ w'\ g\omega\ w\omega$ **by**(*rule wave-restrict-current*)
have *d-IN* $(g\omega \upharpoonleft \Gamma'\ n\ /\ g\ n)\ b = 0$
by(*rule d-IN-restrict-current-outside roofed-greaterI b-TER*)+
hence *SAT-subset*: *SAT* (*quotient-web* $(\Gamma'\ 0)\ (g'\ n)$) $(g\omega \upharpoonleft \Gamma'\ n\ /\ g\ n) \subseteq$ *SAT* $?\Gamma'\ (g\omega \upharpoonleft \Gamma'\ n\ /\ g\ n)$
using *b-TER*[*of n*] *wb-pos*
by(*auto simp add*: *SAT.simps TER-*g' *RF-circ wb-conv* ε*-def field-simps*

ennreal-minus-if split: *if-split-asm*)
hence *TER-subset*: $TER_{quotient\text{-}web\ (\Gamma'\ 0)\ (g'\ n)}\ (g\omega \upharpoonright \Gamma'\ n\ /\ g\ n) \subseteq TER_{?\Gamma'}$ $(g\omega \upharpoonright \Gamma'\ n\ /\ g\ n)$
using *SINK-g'* **by**(*auto simp add*: *restrict-curr-g'*)

show *separating* $?\Gamma'$ ($TER_{?\Gamma'}$ $?g\omega$) (**is** *separating - ?TER*)
proof
fix *x y p*
assume *xy*: $x \in A\ ?\Gamma'\ y \in B\ ?\Gamma'$ **and** *p*: *path* $?\Gamma'$ *x p y*
from *p* **have** *p'*: *path* (*quotient-web* (Γ' *0*) (*g' n*)) *x p y* **by**(*simp add*: *edge-restrict-*Γ')
with *waveD-separating*[*OF* ∗, *THEN separatingD*, *simplified*, *OF p'*] *TER-g'*[*of n*] *SINK-g' SAT-g' restrict-curr-g' SAT-subset xy*
show ($\exists\, z \in set\ p.\ z \in\ ?TER$) $\vee$ $x \in\ ?TER$ **by** *auto*
qed

show *d-OUT* $(g\omega \upharpoonright \Gamma'\ n\ /\ g\ n)\ x = 0$ **if** $x \notin RF_{?\Gamma'}\ ?TER$ **for** *x*
unfolding *restrict-curr-g'*[*symmetric*] **using** *TER-subset that*
by(*intro waveD-OUT*[*OF* ∗])(*auto simp add*: *TER-g' restrict-curr-g' RF-restrict intro*: *in-roofed-mono*)
qed

have *RF-g*ω: *RF* ($TER_{\Gamma'\ 0}\ g\omega$) = *RF* ($\bigcup n.\ TER_{\Gamma'\ 0}\ (g'\ n)$)
proof −
have $RF_{\Gamma'\ 0}$ ($TER_{\Gamma'\ 0}\ g\omega$) = *RF* ($\bigcup i.\ TER_{\Gamma'\ 0}\ (f\ i)$)
unfolding *g*ω*-def* **by**(*subst RF-TER-Sup*[*OF - - chain-f*])(*auto simp add*: *f fw supp-f*)
also have ... = *RF* ($\bigcup i.$ *RF* ($TER_{\Gamma'\ 0}\ (f\ i)$)) **by**(*simp add*: *roofed-UN*)
also have ... = *RF* ($\bigcup i.\ \bigcup j{\le}i.\ TER_{\Gamma'\ 0}\ (g'\ j)$) **unfolding** *RF-f roofed-UN* **by** *simp*
also have ($\bigcup i.\ \bigcup j{\le}i.\ TER_{\Gamma'\ 0}\ (g'\ j)$) = ($\bigcup i.\ TER_{\Gamma'\ 0}\ (g'\ i)$) **by** *auto*
finally show *?thesis* **by** *simp*
qed

have *SAT-plus-*ω: *SAT* (Γ' *n*) ($g\ n \frown_{\Gamma'\ n} g\omega$) = *SAT* ($\Gamma'$ *0*) ($g'\ n \frown_{\Gamma'\ 0} g\omega$) **for** *n*
apply(*intro set-eqI*)
apply(*simp add*: *SAT.simps IN-plus-current*[*OF g w g*ω*r*] *IN-plus-current*[*OF g' w' g*ω*r'*] *TER-g'*)
apply(*cases d-IN* ($g\omega \upharpoonright \Gamma'\ n\ /\ g\ n$) *b*)
apply(*auto simp add*: *SAT.simps wb-conv d-IN-plus-web IN-g'*)
apply(*simp-all add*: *wb-conv IN-g-b restrict-curr-g'* ε*-def field-simps*)
apply(*metis TER-g' b-TER roofed-greaterI*)+
done
have *SINK-plus-*ω: *SINK* ($g\ n \frown_{\Gamma'\ n} g\omega$) = *SINK* ($g'\ n \frown_{\Gamma'\ 0} g\omega$) **for** *n*
apply(*rule set-eqI*; *simp add*: *SINK.simps OUT-plus-current*[*OF g w g*ω*r*] *OUT-plus-current*[*OF g' w'*] *current-restrict-current*[*OF w' g*ω])
using *factor-pos*[*of n*]
by(*auto simp add*: *RF-circ TER-g' restrict-curr-g' currentD-OUT-eq-0*[*OF g*]

currentD-OUT-eq-0[*OF g'*] *g'-simps split*: *if-split-asm*)
have *TER-plus-ω*: $TER_{\Gamma'\ n}$ ($g\ n \frown_{\Gamma'\ n} g\omega$) = $TER_{\Gamma'\ 0}$ ($g'\ n \frown_{\Gamma'\ 0} g\omega$) **for** n
by(*rule set-eqI iffI*)+(*simp-all add*: *SAT-plus-ω SINK-plus-ω*)

define h **where** $h\ n = g\ n \frown_{\Gamma'\ n} g\omega$ **for** n
have *h*: *current* (Γ' n) (h n) **for** n **unfolding** *h-def* **using** *g w*
by(*rule current-plus-current*)(*rule current-restrict-current*[*OF w gω'*])
have *hw*: *wave* (Γ' n) (h n) **for** n **unfolding** *h-def* **using** *g w gω' wωr* **by**(*rule wave-plus*)

define T **where** $T = TER_{\Gamma'\ 0}\ g\omega$
have *RF-h*: *RF* ($TER_{\Gamma'\ n}$ (h n)) = *RF* T **for** n
proof −
have $RF_{\Gamma'\ 0}$ ($TER_{\Gamma'\ n}$ (h n)) = $RF_{\Gamma'\ 0}$ ($RF_{\Gamma'\ 0}$ ($TER_{\Gamma'\ 0}$ $g\omega$) ∪ $TER_{\Gamma'\ 0}$ (g' n))
unfolding *h-def TER-plus-ω RF-TER-plus-web*[*OF g' w' gω wω*] *roofed-idem-Un1* **by**(*simp add*: *Un-commute*)
also have ... = *RF* (($\bigcup n$. $TER_{\Gamma'\ 0}$ (g' n)) ∪ $TER_{\Gamma'\ 0}$ (g' n))
by(*simp add*: *RF-gω roofed-idem-Un1*)
also have ... = $RF_{\Gamma'\ 0}$ T **unfolding** *T-def*
by(*auto simp add*: *RF-gω intro*!: *arg-cong2*[**where** *f=roofed*] *del*: *equalityI*) *auto*
finally show *?thesis* **by** *simp*
qed
have *OUT-h-nT*: *d-OUT* (h n) x = *0* **if** $x \notin$ *RF* T **for** n x
by(*rule waveD-OUT*[*OF hw*])(*simp add*: *RF-h that*)
have *IN-h-nT*: *d-IN* (h n) x = *0* **if** $x \notin$ *RF* T **for** n x
by(*rule wave-not-RF-IN-zero*[*OF h hw*])(*simp add*: *RF-h that*)
have *OUT-h-b*: *d-OUT* (h n) b = *0* **for** n **using** *b-TER*[*of n*] *b-SINK-gω*[*THEN in-SINK-restrict-current*]
by(*auto simp add*: *h-def OUT-plus-current*[*OF g w gωr*] *SINK.simps*)
have *OUT-h-ℰ*: *d-OUT* (h n) x = *0* **if** $x \in \mathcal{E}$ T **for** x n **using** *that*
apply(*subst* (*asm*) *ℰ-RF*[*symmetric*])
apply(*subst* (*asm*) (*1 2*) *RF-h*[*symmetric*, *of n*])
apply(*subst* (*asm*) *ℰ-RF*)
apply(*simp add*: *SINK.simps*)
done
have *IN-h-ℰ*: *d-IN* (h n) x = *weight* (Γ' n) x **if** $x \in \mathcal{E}$ T $x \notin A$ Γ **for** x n **using** *that*
apply(*subst* (*asm*) *ℰ-RF*[*symmetric*])
apply(*subst* (*asm*) (*1 2*) *RF-h*[*symmetric*, *of n*])
apply(*subst* (*asm*) *ℰ-RF*)
apply(*simp add*: *currentD-SAT*[*OF h*])
done

have *b-SAT*: $b \in$ *SAT* (Γ' *0*) (h *0*) **using** *b-TER*[*of 0*]
by(*auto simp add*: *h-def SAT.simps d-IN-plus-web intro*: *order-trans*)
have *b-T*: $b \in T$ **using** *b-SINK-gω b-TER* **by**(*simp add*: *T-def*)(*metis SAT-gω subsetD f-simps*(*1*))

have *essential*: $b \in \mathcal{E}\ T$
proof(*rule ccontr*)
assume $b \notin \mathcal{E}\ T$
hence *b*: $b \notin \mathcal{E}\ (TER_{\Gamma'\ 0}\ (h\ 0))$
proof(*rule contrapos-nn*)
assume $b \in \mathcal{E}\ (TER_{\Gamma'\ 0}\ (h\ 0))$
then obtain $p\ y$ **where** *p*: *path* $\Gamma\ b\ p\ y$ **and** *y*: $y \in B\ \Gamma$ **and** *distinct*: *distinct* $(b \# p)$
and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF\ (TER_{\Gamma'\ 0}\ (h\ 0))$ **by**(*rule* $\mathcal{E}$*-E-RF*) *auto*
from *bypass* **have** *bypass′*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin T$ **unfolding** *RF-h* **by**(*auto intro*: *roofed-greaterI*)
have *essential* $\Gamma\ (B\ \Gamma)\ T\ b$ **using** $p\ y$ **by**(*rule essentialI*)(*auto dest*: *bypass′*)
then show $b \in \mathcal{E}\ T$ **using** *b-T* **by** *simp*
qed

have *h0*: *current* $\Gamma\ (h\ 0)$ **using** $h[of\ 0]$ **by**(*rule current-weight-mono*)(*simp-all add*: *wb-conv* ε*-nonneg*)
moreover have *wave* $\Gamma\ (h\ 0)$
proof
have *separating* $(\Gamma'\ 0)\ (\mathcal{E}_{\Gamma'\ 0}\ (TER_{\Gamma'\ 0}\ (h\ 0)))$ **by**(*rule separating-essential*)(*rule waveD-separating*[*OF hw*])
then have *separating* $\Gamma\ (\mathcal{E}\ (TER_{\Gamma'\ 0}\ (h\ 0)))$ **by**(*simp add*: *separating-gen.simps*)
moreover have *subset*: $\mathcal{E}\ (TER_{\Gamma'\ 0}\ (h\ 0)) \subseteq TER\ (h\ 0)$ **using** ε*-nonneg*[*of* *0*] *b*
by(*auto simp add*: *SAT.simps wb-conv split*: *if-split-asm*)
ultimately show *separating* $\Gamma\ (TER\ (h\ 0))$ **by**(*rule separating-weakening*)
qed(*rule h0*)
ultimately have $h\ 0 = zero\text{-}current$ **by**(*rule looseD-wave*[*OF loose*])
then have $d\text{-}IN\ (h\ 0)\ b = 0$ **by**(*simp*)
with *b-SAT wb* ‹$b \notin A\ \Gamma$› **show** *False* **by**(*simp add*: *SAT.simps wb-conv* ε*-def ennreal-minus-if split*: *if-split-asm*)
qed

define *S* **where** $S = \{x \in RF\ (T \cap B\ \Gamma) \cap A\ \Gamma.\ essential\ \Gamma\ (T \cap B\ \Gamma)\ (RF\ (T \cap B\ \Gamma) \cap A\ \Gamma)\ x\}$
define Γ*-h* **where** $\Gamma\text{-}h = (\!|\ edge = \lambda x\ y.\ edge\ \Gamma\ x\ y \wedge x \in S \wedge y \in T \wedge y \in B\ \Gamma,\ weight = \lambda x.\ weight\ \Gamma\ x * indicator\ (S \cup T \cap B\ \Gamma)\ x,\ A = S,\ B = T \cap B\ \Gamma |\!)$
have Γ*-h-sel* [*simp*]:
$edge\ \Gamma\text{-}h\ x\ y \longleftrightarrow edge\ \Gamma\ x\ y \wedge x \in S \wedge y \in T \wedge y \in B\ \Gamma$
$A\ \Gamma\text{-}h = S$
$B\ \Gamma\text{-}h = T \cap B\ \Gamma$
$weight\ \Gamma\text{-}h\ x = weight\ \Gamma\ x * indicator\ (S \cup T \cap B\ \Gamma)\ x$
for $x\ y$
by(*simp-all add*: Γ*-h-def*)

have *vertex-*Γ*-hD*: $x \in S \cup (T \cap B\ \Gamma)$ **if** *vertex* $\Gamma\text{-}h\ x$ **for** x
using *that* **by**(*auto simp add*: *vertex-def*)

have *S-vertex*: *vertex* Γ-*h* x **if** $x \in S$ **for** x
proof −
from *that* **have** *a*: $x \in A\ \Gamma$ **and** *RF*: $x \in RF\ (T \cap B\ \Gamma)$ **and** *ess*: *essential* Γ $(T \cap B\ \Gamma)\ (RF\ (T \cap B\ \Gamma) \cap A\ \Gamma)\ x$
by(*simp-all add*: *S-def*)
from *ess* **obtain** $p\ y$ **where** *p*: *path* $\Gamma\ x\ p\ y$ **and** *y*: $y \in B\ \Gamma$ **and** *yT*: $y \in T$ **and** *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF\ (T \cap B\ \Gamma) \cap A\ \Gamma$ **by**(*rule essentialE-RF*)(*auto intro*: *roofed-greaterI*)
from $p\ a\ y$ *disjoint* **have** *edge* $\Gamma\ x\ y$
by(*cases*)(*auto 4 3 elim*: *rtrancl-path.cases dest*: *bipartite-E*)
with *that* y *yT* **show** *?thesis* **by**(*auto intro*!: *vertexI1*)
qed
have *OUT-not-S*: *d-OUT* $(h\ n)\ x = 0$ **if** $x \notin S$ **for** $x\ n$
proof(*rule classical*)
assume $*$: *d-OUT* $(h\ n)\ x \neq 0$
consider $(A)\ x \in A\ \Gamma \mid (B)\ x \in B\ \Gamma \mid (outside)\ x \notin A\ \Gamma\ x \notin B\ \Gamma$ **by** *blast*
then show *?thesis*
proof *cases*
case B **with** *currentD-OUT*[*OF h*, *of x n*] **show** *?thesis* **by** *simp*
next
case *outside* **with** *currentD-outside-OUT*[*OF h*, *of x n*] **show** *?thesis* **by**(*simp add*: *not-vertex*)
next
case A
from $*$ **obtain** y **where** *xy*: $h\ n\ (x, y) \neq 0$ **using** *currentD-OUT-eq-0*[*OF h*, *of n x*] **by** *auto*
then have *edge*: *edge* $\Gamma\ x\ y$ **using** *currentD-outside*[*OF h*] **by**(*auto*)
hence *p*: *path* $\Gamma\ x\ [y]\ y$ **by**(*simp add*: *rtrancl-path-simps*)

from *bipartite-E*[*OF edge*] **have** *x*: $x \in A\ \Gamma$ **and** *y*: $y \in B\ \Gamma$ **by** *simp-all*
moreover have $x \in RF\ (RF\ (T \cap B\ \Gamma))$
proof
fix $p\ y'$
assume *p*: *path* $\Gamma\ x\ p\ y'$ **and** y': $y' \in B\ \Gamma$
from $p\ x\ y'$ *disjoint* **have** *py*: $p = [y']$
by(*cases*)(*auto 4 3 elim*: *rtrancl-path.cases dest*: *bipartite-E*)
have *separating* $(\Gamma'\ 0)\ (RF_{\Gamma'\ 0}\ (TER_{\Gamma'\ 0}\ (h\ 0)))$ **unfolding** *separating-RF*
by(*rule waveD-separating*[*OF hw*])
from *separatingD*[*OF this*, *of x p y'*] *py p x y'*
have $x \in RF\ T \vee y' \in RF\ T$ **by**(*auto simp add*: *RF-h*)
thus $(\exists z{\in}set\ p.\ z \in RF\ (T \cap B\ \Gamma)) \vee x \in RF\ (T \cap B\ \Gamma)$
proof *cases*
case *right* **with** y' *py* **show** *?thesis* **by**(*simp add*: *RF-in-B*)
next
case *left*
have $x \notin \mathcal{E}\ T$ **using** *OUT-h-*$\mathcal{E}$[*of x n*] *xy* **by**(*auto simp add*: *currentD-OUT-eq-0*[*OF h*])
with *left* **have** $x \in RF^{\circ}\ T$ **by**(*simp add*: *roofed-circ-def*)
from *RF-circ-edge-forward*[*OF this*, *of y'*] *p py* **have** $y' \in RF\ T$ **by**(*simp*

add: *rtrancl-path-simps*)
with y' **have** $y' \in T$ **by**(*simp add*: *RF-in-B*)
with y' **show** *?thesis* **using** *py* **by**(*auto intro*: *roofed-greaterI*)
qed
qed
moreover have $y \in T$ **using** *IN-h-nT*[*of y n*] *y xy* **by**(*auto simp add*: *RF-in-B currentD-IN-eq-0*[*OF h*])
with *p x y disjoint* **have** *essential* Γ ($T \cap B\ \Gamma$) (*RF* ($T \cap B\ \Gamma$) $\cap A\ \Gamma$) x **by**(*auto intro*!: *essentialI*)
ultimately have $x \in S$ **unfolding** *roofed-idem* **by**(*simp add*: *S-def*)
with *that* **show** *?thesis* **by** *contradiction*
qed
qed

have *B-vertex*: *vertex* Γ-*h* y **if** T: $y \in T$ **and** B: $y \in B\ \Gamma$ **and** w: *weight* $\Gamma\ y > 0$ **for** y
proof $-$
from *T B disjoint* ε-*less*[*of 0*] w
have *d-IN* ($h\ 0$) $y > 0$ **using** *IN-h-*$\mathcal{E}$[*of y 0*] **by**(*cases* $y \in A\ \Gamma$)(*auto simp add*: *essential-BI wb-conv ennreal-minus-if*)
then obtain x **where** *xy*: $h\ 0\ (x, y) \neq 0$ **using** *currentD-IN-eq-0*[*OF h*, *of 0 y*] **by**(*auto*)
then have *edge*: *edge* $\Gamma\ x\ y$ **using** *currentD-outside*[*OF h*] **by**(*auto*)
from *xy* **have** *d-OUT* ($h\ 0$) $x \neq 0$ **by**(*auto simp add*: *currentD-OUT-eq-0*[*OF h*])
hence $x \in S$ **using** *OUT-not-S*[*of x 0*] **by**(*auto*)
with *edge T B* **show** *?thesis* **by**(*simp add*: *vertexI2*)
qed

have Γ-*h*: *countable-bipartite-web* Γ-*h*
proof
show $\mathbf{V}_{\Gamma\text{-}h} \subseteq A\ \Gamma\text{-}h \cup B\ \Gamma\text{-}h$ **by**(*auto simp add*: *vertex-def*)
show $A\ \Gamma\text{-}h \subseteq \mathbf{V}_{\Gamma\text{-}h}$ **using** *S-vertex* **by** *auto*
show $x \in A\ \Gamma\text{-}h \wedge y \in B\ \Gamma\text{-}h$ **if** *edge* Γ-*h* $x\ y$ **for** $x\ y$ **using** *that* **by** *auto*
show $A\ \Gamma\text{-}h \cap B\ \Gamma\text{-}h = \{\}$ **using** *disjoint* **by**(*auto simp add*: *S-def*)
have $\mathbf{E}_{\Gamma\text{-}h} \subseteq \mathbf{E}$ **by** *auto*
thus *countable* $\mathbf{E}_{\Gamma\text{-}h}$ **by**(*rule countable-subset*) *simp*
show *weight* Γ-*h* $x \neq \top$ **for** x **by**(*simp split*: *split-indicator*)
show *weight* Γ-*h* $x = 0$ **if** $x \notin \mathbf{V}_{\Gamma\text{-}h}$ **for** x
using *that S-vertex B-vertex*[*of x*]
by(*cases weight* Γ-*h* $x > 0$)(*auto split*: *split-indicator*)
qed
then interpret Γ-*h*: *countable-bipartite-web* Γ-*h* **.**

have *essential-T*: *essential* Γ ($B\ \Gamma$) T = *essential* Γ ($B\ \Gamma$) ($TER_{\Gamma'\ 0}$ ($h\ 0$))
proof(*rule ext iffI*)+
fix x
assume *essential* Γ ($B\ \Gamma$) $T\ x$
then obtain $p\ y$ **where** p: *path* $\Gamma\ x\ p\ y$ **and** y: $y \in B\ \Gamma$ **and** *distinct*: *distinct*

(x # p)
    and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF\ T$ **by**(*rule essentialE-RF*)*auto*
  from *bypass* **have** *bypass'*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin TER_{\Gamma'\ 0}\ (h\ 0)$
    unfolding *RF-h*[*of 0*, *symmetric*] **by**(*blast intro*: *roofed-greaterI*)
  show *essential* Γ (B Γ) ($TER_{\Gamma'\ 0}$ (h 0)) x **using** p y
    by(*blast intro*: *essentialI dest*: *bypass'*)
next
  fix x
  assume *essential* Γ (B Γ) ($TER_{\Gamma'\ 0}$ (h 0)) x
  then obtain p y **where** p: *path* Γ x p y **and** y: $y \in B\ \Gamma$ **and** *distinct*: *distinct* (x # p)
    and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF\ (TER_{\Gamma'\ 0}\ (h\ 0))$ **by**(*rule essentialE-RF*)*auto*
  from *bypass* **have** *bypass'*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin T$
    unfolding *RF-h*[*of 0*] **by**(*blast intro*: *roofed-greaterI*)
  show *essential* Γ (B Γ) T x **using** p y
    by(*blast intro*: *essentialI dest*: *bypass'*)
qed

have h': *current* Γ-h (h n) **for** n
proof
  show *d-OUT* (h n) $x \leq$ *weight* Γ-h x **for** x
   using *currentD-weight-OUT*[*OF h*, *of n x*] ε*-nonneg*[*of n*] Γ'.*currentD-OUT'*[*OF h*, *of x n*] *OUT-not-S*
   by(*auto split*: *split-indicator if-split-asm elim*: *order-trans intro*: *diff-le-self-ennreal in-roofed-mono simp add*: *OUT-h-b roofed-circ-def*)

  show *d-IN* (h n) $x \leq$ *weight* Γ-h x **for** x
   using *currentD-weight-IN*[*OF h*, *of n x*] *currentD-IN*[*OF h*, *of x n*] ε*-nonneg*[*of n*] *b-T b* Γ'.*currentD-IN'*[*OF h*, *of x n*] *IN-h-nT*[*of x n*]
    by(*cases* $x \in B\ \Gamma$)(*auto 4 3 split*: *split-indicator split*: *if-split-asm elim*: *order-trans intro*: *diff-le-self-ennreal simp add*: *S-def roofed-circ-def RF-in-B*)

  show h n e = 0 **if** $e \notin \mathbf{E}_{\Gamma\text{-}h}$ **for** e
   using *that OUT-not-S*[*of fst e n*] *currentD-outside'*[*OF h*, *of e n*] Γ'.*currentD-IN'*[*OF h*, *of snd e n*] *disjoint*
    apply(*cases* $e \in \mathbf{E}$)
     apply(*auto split*: *prod.split-asm simp add*: *currentD-OUT-eq-0*[*OF h*] *currentD-IN-eq-0*[*OF h*])
    apply(*cases fst* $e \in S$; *clarsimp simp add*: *S-def*)
    apply(*frule RF-circ-edge-forward*[*rotated*])
    apply(*erule roofed-circI*, *blast*)
    apply(*drule bipartite-E*)
    apply(*simp add*: *RF-in-B*)
    done
qed

have *SAT-h'*: B Γ-h $\cap$ $\mathbf{V}_{\Gamma\text{-}h}$ $-$ $\{b\}$ $\subseteq$ *SAT* Γ-h (h n) **for** n
proof

fix x
assume $x \in B\ \Gamma\text{-}h \cap \mathbf{V}_{\Gamma\text{-}h} - \{b\}$
then have x: $x \in T$ **and** B: $x \in B\ \Gamma$ **and** b: $x \neq b$ **and** *vertex*: $x \in \mathbf{V}_{\Gamma\text{-}h}$ **by** *auto*
from *B disjoint* **have** *xnA*: $x \notin A\ \Gamma$ **by** *blast*
from *x B* **have** $x \in \mathcal{E}\ T$ **by**(*simp add*: *essential-BI*)
hence *d-IN* $(h\ n)\ x$ = *weight* $(\Gamma'\ n)\ x$ **using** *xnA* **by**(*rule IN-h-$\mathcal{E}$*)
with *xnA b x B* **show** $x \in$ *SAT* $\Gamma\text{-}h\ (h\ n)$ **by**(*simp add*: *currentD-SAT*[*OF h'*])
qed
moreover have $b \in B\ \Gamma\text{-}h$ **using** *b essential* **by** *simp*
moreover have $(\lambda n.\ min\ \delta\ wb * (1\ /\ (real\ (n + 2)))) \longrightarrow 0$
apply(*rule LIMSEQ-ignore-initial-segment*)
apply(*rule tendsto-mult-right-zero*)
apply(*rule lim-1-over-real-power*[**where** *s=1*, *simplified*])
done
then have (*INF n. ennreal* $(\varepsilon\ n)) = 0$ **using** *wb-pos* δ
apply(*simp add*: *ε-def*)
apply(*rule INF-Lim*)
apply(*rule decseq-SucI*)
apply(*simp add*: *field-simps min-def*)
apply(*simp add*: *add.commute ennreal-0*[*symmetric*] *del*: *ennreal-0*)
done
then have (*SUP n. d-IN* $(h\ n)\ b)$ = *weight* $\Gamma\text{-}h\ b$ **using** *essential b bnA wb IN-h-$\mathcal{E}$*[*of b*]
by(*simp add*: *SUP-const-minus-ennreal*)
moreover have *d-IN* $(h\ 0)\ b \le$ *d-IN* $(h\ n)\ b$ **for** n **using** *essential b bnA wb-pos* δ *IN-h-$\mathcal{E}$*[*of b*]
by(*simp add*: *wb-conv ε-def field-simps ennreal-minus-if min-le-iff-disj*)
moreover have *b-V*: $b \in \mathbf{V}_{\Gamma\text{-}h}$ **using** *b wb essential* **by**(*auto dest*: *B-vertex*)
ultimately have $\exists h'.$ *current* $\Gamma\text{-}h\ h' \wedge$ *wave* $\Gamma\text{-}h\ h' \wedge B\ \Gamma\text{-}h \cap \mathbf{V}_{\Gamma\text{-}h} \subseteq$ *SAT* $\Gamma\text{-}h\ h'$
by(*rule* Γ-*h.unhinder-bipartite*[*OF h'*])
then obtain h' **where** h': *current* $\Gamma\text{-}h\ h'$ **and** $h'w$: *wave* $\Gamma\text{-}h\ h'$
and *B-SAT'*: $B\ \Gamma\text{-}h \cap \mathbf{V}_{\Gamma\text{-}h} \subseteq$ *SAT* $\Gamma\text{-}h\ h'$ **by** *blast*

have h'': *current* $\Gamma\ h'$
proof
show *d-OUT* $h'\ x \le$ *weight* $\Gamma\ x$ **for** x **using** *currentD-weight-OUT*[*OF h'*, *of x*]
by(*auto split*: *split-indicator-asm elim*: *order-trans intro*:)
show *d-IN* $h'\ x \le$ *weight* $\Gamma\ x$ **for** x **using** *currentD-weight-IN*[*OF h'*, *of x*]
by(*auto split*: *split-indicator-asm elim*: *order-trans intro*:)
show $h'\ e = 0$ **if** $e \notin \mathbf{E}$ **for** e **using** *currentD-outside'*[*OF h'*, *of e*] *that* **by** *auto*
qed
moreover have *wave* $\Gamma\ h'$
proof
have *separating* $(\Gamma'\ 0)\ T$ **unfolding** *T-def* **by**(*rule waveD-separating*[*OF wω*])
hence *separating* $\Gamma\ T$ **by**(*simp add*: *separating-gen.simps*)

```
  hence ∗: separating Γ (ℰ T) by(rule separating-essential)
  show separating Γ (TER h′)
  proof
    fix x p y
    assume x: x ∈ A Γ and p: path Γ x p y and y: y ∈ B Γ
    from p x y disjoint have py: p = [y]
      by(cases)(auto 4 3 elim: rtrancl-path.cases dest: bipartite-E)
    from separatingD[OF ∗ p x y] py have x ∈ ℰ T ∨ y ∈ ℰ T by auto
    then show (∃ z∈set p. z ∈ TER h′) ∨ x ∈ TER h′
    proof cases
      case left
      then have x ∉ V_Γ-h using x disjoint
        by(auto 4 4 dest!: vertex-Γ-hD simp add: S-def elim: essentialE-RF intro!:
roofed-greaterI dest: roofedD)
      hence d-OUT h′ x = 0 by(intro currentD-outside-OUT[OF h′])
      with x have x ∈ TER h′ by(auto simp add: SAT.A SINK.simps)
      thus ?thesis ..
    next
      case right
      have y ∈ SAT Γ h′
      proof(cases weight Γ y > 0)
        case True
        with py x y right have vertex Γ-h y by(auto intro: B-vertex)
        hence y ∈ SAT Γ-h h′ using B-SAT′ right y by auto
        with right y disjoint show ?thesis
          by(auto simp add: currentD-SAT[OF h′] currentD-SAT[OF h″] S-def)
      qed(auto simp add: SAT.simps)
       with currentD-OUT[OF h′, of y] y right have y ∈ TER h′ by(auto simp
add: SINK)
      thus ?thesis using py by simp
    qed
  qed
 qed(rule h″)
 ultimately have h′ = zero-current by(rule looseD-wave[OF loose])
 hence d-IN h′ b = 0 by simp
 moreover from essential b b-V B-SAT′ have b ∈ SAT Γ-h h′ by(auto)
 ultimately show False using wb b essential disjoint by(auto simp add: SAT.simps
S-def)
qed

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* Γ *f* $\longleftrightarrow$ *current* Γ *f*
using *bipartite-V* **by**(*auto simp add*: *web-flow.simps*)

lemma *countable-bipartite-web-minus-web*:
assumes *f*: *current* Γ *f*
shows *countable-bipartite-web* $(\Gamma \ominus f)$
using *bipartite-V A-vertex bipartite-E disjoint currentD-finite-OUT*[*OF f*] *currentD-weight-OUT*[*OF f*] *currentD-weight-IN*[*OF f*] *currentD-outside-OUT*[*OF f*] *currentD-outside-IN*[*OF f*]
by *unfold-locales* (*auto simp add*: *weight-outside*)

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
interpret Γ: *countable-bipartite-web* $\Gamma \ominus f$ **using** *f* **by**(*rule countable-bipartite-web-minus-web*)
show *d-OUT ?fg x* $\leq$ *weight* Γ *x* **for** *x*
using *currentD-weight-OUT*[*OF g*, *of x*] *currentD-OUT*[*OF g*, *of x*] *currentD-finite-OUT*[*OF f*, *of x*] *currentD-OUT*[*OF f*, *of x*] *currentD-outside-IN*[*OF f*, *of x*] *currentD-outside-OUT*[*OF f*, *of x*] *currentD-weight-OUT*[*OF f*, *of x*]
by(*cases* $x \in A\ \Gamma \vee x \in B\ \Gamma$)(*auto simp add*: *add.commute d-OUT-def nn-integral-add not-vertex ennreal-le-minus-iff split*: *if-split-asm*)
show *d-IN ?fg x* $\leq$ *weight* Γ *x* **for** *x*
using *currentD-weight-IN*[*OF g*, *of x*] *currentD-IN*[*OF g*, *of x*] *currentD-finite-IN*[*OF f*, *of x*] *currentD-OUT*[*OF f*, *of x*] *currentD-outside-IN*[*OF f*, *of x*] *currentD-outside-OUT*[*OF f*, *of x*] *currentD-weight-IN*[*OF f*, *of x*]
by(*cases* $x \in A\ \Gamma \vee x \in B\ \Gamma$)(*auto simp add*: *add.commute d-IN-def nn-integral-add not-vertex ennreal-le-minus-iff split*: *if-split-asm*)
show *?fg e* $= 0$ **if** $e \notin$ **E** **for** *e* **using** *that currentD-outside'*[*OF f*, *of e*] *currentD-outside'*[*OF g*, *of e*] **by**(*cases e*) *simp*
qed

lemma *wave-plus-current-minus*:
assumes *f*: *current* Γ *f*
and *w*: *wave* Γ *f*

and *g*: *current* (Γ ⊖ *f*) *g*
and *w'*: *wave* (Γ ⊖ *f*) *g*
shows *wave* Γ (*plus-current f g*) (**is** *wave - ?fg*)
proof
show *fg*: *current* Γ *?fg* **using** *f g* **by**(*rule current-plus-current-minus*)
show *sep*: *separating* Γ (*TER ?fg*)
proof
fix *x p y*
assume *x*: *x* ∈ *A* Γ **and** *p*: *path* Γ *x p y* **and** *y*: *y* ∈ *B* Γ
from *p x y disjoint* **have** *py*: *p* = [*y*]
by(*cases*)(*auto 4 3 elim*: *rtrancl-path.cases dest*: *bipartite-E*)
with *waveD-separating*[*THEN separatingD*, *OF w p x y*] **have** *x* ∈ *TER f* ∨ *y* ∈ *TER f* **by** *auto*
thus (∃ *z*∈*set p*. *z* ∈ *TER ?fg*) ∨ *x* ∈ *TER ?fg*
proof *cases*
case *right*
with *y disjoint* **have** *y* ∈ *TER ?fg* **using** *currentD-OUT*[*OF fg y*]
by(*auto simp add*: *SAT.simps SINK.simps d-IN-def nn-integral-add not-le add-increasing2*)
thus *?thesis* **using** *py* **by** *simp*
next
case *x'*: *left*
from *p* **have** *path* (Γ ⊖ *f*) *x p y* **by** *simp*
from *waveD-separating*[*THEN separatingD*, *OF w' this*] *x y py*
have $x \in TER_{\Gamma \ominus f}\ g \lor y \in TER_{\Gamma \ominus f}\ g$ **by** *auto*
thus *?thesis*
proof *cases*
case *left*
hence *x* ∈ *TER ?fg* **using** *x x'*
by(*auto simp add*: *SAT.simps SINK.simps d-OUT-def nn-integral-add*)
thus *?thesis* **..**
next
case *right*
hence *y* ∈ *TER ?fg* **using** *disjoint y currentD-OUT*[*OF fg y*] *currentD-OUT*[*OF f y*] *currentD-finite-IN*[*OF f*, *of y*]
by(*auto simp add*: *add.commute SINK.simps SAT.simps d-IN-def nn-integral-add ennreal-minus-le-iff split*: *if-split-asm*)
with *py* **show** *?thesis* **by** *auto*
qed
qed
qed
qed

lemma *minus-plus-current*:
assumes *f*: *current* Γ *f*
and *g*: *current* (Γ ⊖ *f*) *g*
shows Γ ⊖ *plus-current f g* = Γ ⊖ *f* ⊖ *g* (**is** *?lhs* = *?rhs*)
proof(*rule web.equality*)
have *weight ?lhs x* = *weight ?rhs x* **for** *x*

using *currentD-weight-IN*[*OF f*, *of x*] *currentD-weight-IN*[*OF g*, *of x*]
by (*auto simp add*: *d-IN-def d-OUT-def nn-integral-add diff-add-eq-diff-diff-swap-ennreal add-increasing2 diff-add-assoc2-ennreal add.assoc*)
thus *weight ?lhs* = *weight ?rhs* **..**
qed *simp-all*

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
assume *hindered* ($\Gamma \ominus f$)
then obtain *g* **where** *g*: *current* ($\Gamma \ominus f$) *g*
and *w'*: *wave* ($\Gamma \ominus f$) *g*
and *hind*: *hindrance* ($\Gamma \ominus f$) *g* **by** *cases*
let *?fg* = *plus-current f g*
have *fg*: *current* Γ *?fg* **using** *f g* **by**(*rule current-plus-current-minus*)
moreover have *wave* Γ *?fg* **using** *f w g w'* **by**(*rule wave-plus-current-minus*)
moreover from *hind* **obtain** *a* **where** *a*: $a \in A\ \Gamma$ **and** $n\mathcal{E}$: $a \notin \mathcal{E}_{\Gamma \ominus f}$ ($TER_{\Gamma \ominus f}$ *g*)
and *wa*: *d-OUT g a* < *weight* ($\Gamma \ominus f$) *a* **by** *cases auto*
from *a* **have** *hindrance* Γ *?fg*
proof
show $a \notin \mathcal{E}$ (*TER ?fg*)
proof
assume $\mathcal{E}$: $a \in \mathcal{E}$ (*TER ?fg*)
then obtain *p y* **where** *p*: *path* Γ *a p y* **and** *y*: $y \in B\ \Gamma$
and *bypass*: $\bigwedge z.\ z \in$ *set p* $\Longrightarrow z \notin$ *RF* (*TER ?fg*) **by**(*rule* $\mathcal{E}$*-E-RF*) *blast*
from *p a y disjoint* **have** *py*: *p* = [*y*]
by(*cases*)(*auto 4 3 elim*: *rtrancl-path.cases dest*: *bipartite-E*)
from *bypass*[*of y*] *py* **have** $y \notin$ *TER ?fg* **by**(*auto intro*: *roofed-greaterI*)
with *currentD-OUT*[*OF fg y*] **have** $y \notin$ *SAT* Γ *?fg* **by**(*auto simp add*: *SINK.simps*)
hence $y \notin$ *SAT* ($\Gamma \ominus f$) *g* **using** *y currentD-OUT*[*OF f y*] *currentD-finite-IN*[*OF f*, *of y*]
by(*auto simp add*: *SAT.simps d-IN-def nn-integral-add ennreal-minus-le-iff add.commute*)
hence *essential* ($\Gamma \ominus f$) (*B* ($\Gamma \ominus f$)) ($TER_{\Gamma \ominus f}$ *g*) *a* **using** *p py y*
by(*auto intro*!: *essentialI*)
moreover from $\mathcal{E}$ *a* **have** $a \in TER_{\Gamma \ominus f}$ *g*
by(*auto simp add*: *SAT.A SINK-plus-current*)
ultimately have $a \in \mathcal{E}_{\Gamma \ominus f}$ ($TER_{\Gamma \ominus f}$ *g*) **by** *blast*
thus *False* **using** $n\mathcal{E}$ **by** *contradiction*
qed
show *d-OUT ?fg a* < *weight* Γ *a* **using** *a wa currentD-finite-OUT*[*OF f*, *of a*]
by(*simp add*: *d-OUT-def less-diff-eq-ennreal less-top add.commute nn-integral-add*)
qed
ultimately have *hindered* Γ **by**(*blast intro*: *hindered.intros*)

with *unhindered* **show** *False* **by** *contradiction*
qed

lemma *loose-minus-web*:
assumes *unhindered*: $\neg$ *hindered* Γ
and *f*: *current* Γ *f*
and *w*: *wave* Γ *f*
and *maximal*: $\bigwedge w.\ [\![\ current\ \Gamma\ w;\ wave\ \Gamma\ w;\ f \leq w\ ]\!] \Longrightarrow f = w$
shows *loose* $(\Gamma \ominus f)$ (**is** *loose* $?\Gamma$)
proof
fix *g*
assume *g*: *current* $?\Gamma$ *g* **and** w': *wave* $?\Gamma$ *g*
let $?g$ = *plus-current f g*
from *f g* **have** *current* Γ $?g$ **by**(*rule current-plus-current-minus*)
moreover from *f w g* w' **have** *wave* Γ $?g$ **by**(*rule wave-plus-current-minus*)
moreover have $f \leq ?g$ **by**(*clarsimp simp add*: *le-fun-def*)
ultimately have *eq*: $f = ?g$ **by**(*rule maximal*)
have $g\ e = 0$ **for** *e*
proof(*cases e*)
case (*Pair x y*)
have $f\ e \leq$ *d-OUT f x* **unfolding** *d-OUT-def Pair* **by**(*rule nn-integral-ge-point*)
simp
also have $\ldots \leq$ *weight* Γ *x* **by**(*rule currentD-weight-OUT*[*OF f*])
also have $\ldots < \top$ **by**(*simp add*: *less-top*[*symmetric*])
finally show $g\ e = 0$ **using** *Pair eq*[*THEN fun-cong*, *of e*]
by(*cases f e g e rule*: *ennreal2-cases*)(*simp-all add*: *fun-eq-iff*)
qed
thus $g = (\lambda\text{-}.\ 0)$ **by**(*simp add*: *fun-eq-iff*)
next
show $\neg$ *hindrance* $?\Gamma$ *zero-current* **using** *unhindered*
proof(*rule contrapos-nn*)
assume *hindrance* $?\Gamma$ *zero-current*
then obtain *x* **where** *a*: $x \in A\ ?\Gamma$ **and** $\mathcal{E}$: $x \notin \mathcal{E}_{?\Gamma}$ (*TER*$_{?\Gamma}$ *zero-current*)
and *weight*: *d-OUT zero-current x* $<$ *weight* $?\Gamma$ *x* **by** *cases*
have *hindrance* Γ *f*
proof
show a': $x \in A\ \Gamma$ **using** *a* **by** *simp*
with *weight* **show** *d-OUT f x* $<$ *weight* Γ *x*
by(*simp add*: *less-diff-eq-ennreal less-top*[*symmetric*] *currentD-finite-OUT*[*OF*
f])
show $x \notin \mathcal{E}$ (*TER f*)
proof
assume $x \in \mathcal{E}$ (*TER f*)
then obtain *p y* **where** *p*: *path* Γ *x p y* **and** *y*: $y \in B\ \Gamma$
and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF$ (*TER f*) **by**(*rule* $\mathcal{E}$*-E-RF*) *auto*
from *p* a' *y disjoint* **have** *py*: $p = [y]$
by(*cases*)(*auto 4 3 elim*: *rtrancl-path.cases dest*: *bipartite-E*)
hence $y \notin$ (*TER f*) **using** *bypass*[*of y*] **by**(*auto intro*: *roofed-greaterI*)
hence *weight* $?\Gamma$ $y > 0$ **using** *currentD-OUT*[*OF f y*] *disjoint y*

by(*auto simp add*: *SINK.simps SAT.simps diff-gr0-ennreal*)
hence $y \notin$ *TER* $_{?\Gamma}$ *zero-current* **using** *y disjoint* **by**(*auto*)
hence *essential* $?\Gamma$ (B $?\Gamma$) (*TER* $_{?\Gamma}$ *zero-current*) x **using** *p y py* **by**(*auto intro*!: *essentialI*)
with *a* **have** $x \in \mathcal{E}_{?\Gamma}$ (*TER* $_{?\Gamma}$ *zero-current*) **by** *simp*
with $\mathcal{E}$ **show** *False* **by** *contradiction*
qed
qed
thus *hindered* Γ **using** *f w* **..**
qed
qed

lemma *weight-minus-web*:
assumes *f*: *current* Γ *f*
shows *weight* ($\Gamma \ominus f$) x = (*if* $x \in A\ \Gamma$ *then weight* $\Gamma\ x$ − *d-OUT f x else weight* $\Gamma\ x$ − *d-IN f x*)
proof(*cases* $x \in B\ \Gamma$)
case *True*
with *currentD-OUT*[*OF f True*] *disjoint* **show** *?thesis* **by** *auto*
next
case *False*
hence *d-IN f x* = *0* *d-OUT f x* = *0* **if** $x \notin A\ \Gamma$
using *currentD-outside-OUT*[*OF f*, *of x*] *currentD-outside-IN*[*OF f*, *of x*] *bipartite-V that* **by** *auto*
then show *?thesis* **by** *simp*
qed

lemma (**in** −) *separating-minus-web* [*simp*]: *separating-gen* ($G \ominus f$) = *separating-gen G*
by(*simp add*: *separating-gen.simps fun-eq-iff*)

lemma *current-minus*:
assumes *f*: *current* Γ *f*
and *g*: *current* Γ *g*
and *le*: $\bigwedge e.\ g\ e \le f\ e$
shows *current* ($\Gamma \ominus g$) ($f - g$)
proof −
interpret Γ: *countable-bipartite-web* $\Gamma \ominus g$ **using** *g* **by**(*rule countable-bipartite-web-minus-web*)
note [*simp del*] = *minus-web-sel*(*2*)
and [*simp*] = *weight-minus-web*[*OF g*]
show *?thesis*
proof
show *d-OUT* ($f - g$) $x \le$ *weight* ($\Gamma \ominus g$) x **for** x **unfolding** *fun-diff-def*
using *currentD-weight-OUT*[*OF f*, *of x*] *currentD-weight-IN*[*OF g*, *of x*]
by(*subst d-OUT-diff*)(*simp-all add*: *le currentD-finite-OUT*[*OF g*] *currentD-OUT'*[*OF f*] *currentD-OUT'*[*OF g*] *ennreal-minus-mono*)
show *d-IN* ($f - g$) $x \le$ *weight* ($\Gamma \ominus g$) x **for** x **unfolding** *fun-diff-def*
using *currentD-weight-IN*[*OF f*, *of x*] *currentD-weight-OUT*[*OF g*, *of x*]

by(*subst d-IN-diff*)(*simp-all add*: *le currentD-finite-IN*[*OF g*] *currentD-IN*[*OF f*] *currentD-IN*[*OF g*] *ennreal-minus-mono*)
show $(f - g)\ e = 0$ **if** $e \notin \mathbf{E}_{\Gamma \ominus g}$ **for** *e* **using** *that currentD-outside′*[*OF f*] *currentD-outside′*[*OF g*] **by** *simp*
qed
qed

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*: $TER\ f \subseteq TER_{\Gamma \ominus g}\ (f - g)$
proof
have $x \in SINK\ f \Longrightarrow x \in SINK\ (f - g)$ **for** *x* **using** *d-OUT-mono*[*of g - f*, *OF le*, *of x*]
by(*auto simp add*: *SINK.simps fun-diff-def d-OUT-diff le currentD-finite-OUT*[*OF g*])
moreover have $x \in SAT\ \Gamma\ f \Longrightarrow x \in SAT\ (\Gamma \ominus g)\ (f - g)$ **for** *x*
by(*auto simp add*: *SAT.simps currentD-OUT′*[*OF g*] *fun-diff-def d-IN-diff le currentD-finite-IN*[*OF g*] *ennreal-minus-mono*)
ultimately show *TER*: $TER\ f \subseteq TER_{\Gamma \ominus g}\ (f - g)$ **by**(*auto*)

from *w* **have** *separating* Γ $(TER\ f)$ **by**(*rule waveD-separating*)
thus *separating* $(\Gamma \ominus g)\ (TER_{\Gamma \ominus g}\ (f - g))$ **using** *TER* **by**(*simp add*: *separating-weakening*)

fix *x*
assume $x \notin RF_{\Gamma \ominus g}\ (TER_{\Gamma \ominus g}\ (f - g))$
hence $x \notin RF\ (TER\ f)$ **using** *TER* **by**(*auto intro*: *in-roofed-mono*)
hence $d\text{-}OUT\ f\ x = 0$ **by**(*rule waveD-OUT*[*OF w*])
moreover have $0 \leq f\ e$ **for** *e* **using** *le*[*of e*] **by** *simp*
ultimately show $d\text{-}OUT\ (f - g)\ x = 0$ **unfolding** *d-OUT-def*
by(*simp add*: *nn-integral-0-iff emeasure-count-space-eq-0*)
qed

lemma (**in** −) *essential-minus-web* [*simp*]: *essential* $(\Gamma \ominus f)$ = *essential* Γ
by(*simp add*: *essential-def fun-eq-iff*)

lemma (**in** −) *RF-in-essential*: **fixes** *B* **shows** *essential* $\Gamma\ B\ S\ x \Longrightarrow x \in$ *roofed-gen* $\Gamma\ B\ S \longleftrightarrow x \in S$
by(*auto intro*: *roofed-greaterI elim*!: *essentialE-RF dest*: *roofedD*)

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

case *True*
have *?lhs* = ($\sum^+$ y'. f (x, y') + k ∗ *indicator* {y} y') − ($\sum^+$ y'. f (x, y') ∗ *indicator* {y} y')
unfolding *d-OUT-def* **using** *assms True*
by(*subst nn-integral-diff*[*symmetric*])
(*auto intro*!: *nn-integral-cong simp add*: *AE-count-space split*: *split-indicator*)
also have ($\sum^+$ y'. f (x, y') + k ∗ *indicator* {y} y') = *d-OUT f x* + ($\sum^+$ y'. k ∗ *indicator* {y} y')
unfolding *d-OUT-def* **using** *assms*
by(*subst nn-integral-add*[*symmetric*])
(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... − ($\sum^+$ y'. f (x, y') ∗ *indicator* {y} y') = *?rhs* **using** *True assms*
by(*subst diff-add-assoc2-ennreal*[*symmetric*])(*auto simp add*: *d-OUT-def intro*!: *nn-integral-ge-point*)
finally show *?thesis* **.**
qed(*simp add*: *d-OUT-def*)

lemma *unhindered-saturate1*: — Lemma 6.10
assumes *unhindered*: ¬ *hindered* Γ
and *a*: $a \in A$ Γ
shows ∃f. *current* Γ f ∧ *d-OUT f a* = *weight* Γ a ∧ ¬ *hindered* (Γ ⊖ f)
proof −
from *a A-vertex* **have** *a-vertex*: *vertex* Γ a **by** *auto*
from *unhindered* **have** ¬ *hindrance* Γ *zero-current* **by**(*auto intro*!: *hindered.intros simp add*:)
then have *a-$\mathcal{E}$*: $a \in \mathcal{E}$ (A Γ) **if** *weight* Γ $a > 0$
proof(*rule contrapos-np*)
assume $a \notin \mathcal{E}$ (A Γ)
with a **have** ¬ *essential* Γ (B Γ) (A Γ) a **by** *simp*
hence ¬ *essential* Γ (B Γ) (A Γ ∪ {x. *weight* Γ $x \leq 0$}) a
by(*rule contrapos-nn*)(*erule essential-mono*; *simp*)
with *a that* **show** *hindrance* Γ *zero-current* **by**(*auto intro*: *hindrance*)
qed

define F **where** F = (λ(ε, h :: ′*v current*). *plus-current* ε h)
have *F-simps*: F (ε, h) = *plus-current* ε h **for** ε h **by**(*simp add*: *F-def*)
define *Fld* **where** *Fld* = {(ε, h).
current Γ ε ∧ (∀ x. $x \neq a \longrightarrow$ *d-OUT* ε x = *0*) ∧
current (Γ ⊖ ε) h ∧ *wave* (Γ ⊖ ε) h ∧
¬ *hindered* (Γ ⊖ F (ε, h))}
define *leq* **where** *leq* = *restrict-rel Fld* {(f, f'). $f \leq f'$}
have *Fld*: *Field leq* = *Fld* **by**(*auto simp add*: *leq-def*)
have *F-I* [*intro?*]: (ε, h) ∈ *Field leq*
if *current* Γ ε **and** $\bigwedge x.\ x \neq a \Longrightarrow$ *d-OUT* ε x = *0*
and *current* (Γ ⊖ ε) h **and** *wave* (Γ ⊖ ε) h
and ¬ *hindered* (Γ ⊖ F (ε, h))
for ε h **using** *that* **by**(*simp add*: *Fld Fld-def*)
have *ε-curr*: *current* Γ ε **if** (ε, h) ∈ *Field leq* **for** ε h **using** *that* **by**(*simp add*: *Fld Fld-def*)

have *OUT-ε*: $\bigwedge x.\ x \neq a \Longrightarrow$ *d-OUT* $\varepsilon\ x = 0$ **if** $(\varepsilon, h) \in$ *Field leq* **for** $\varepsilon\ h$ **using** *that* **by**(*simp add*: *Fld Fld-def*)
have *h*: *current* $(\Gamma \ominus \varepsilon)\ h$ **if** $(\varepsilon, h) \in$ *Field leq* **for** $\varepsilon\ h$ **using** *that* **by**(*simp add*: *Fld Fld-def*)
have *h-w*: *wave* $(\Gamma \ominus \varepsilon)\ h$ **if** $(\varepsilon, h) \in$ *Field leq* **for** $\varepsilon\ h$ **using** *that* **by**(*simp add*: *Fld Fld-def*)
have *unhindered'*: $\neg$ *hindered* $(\Gamma \ominus F\ \varepsilon h)$ **if** $\varepsilon h \in$ *Field leq* **for** εh **using** *that* **by**(*simp add*: *Fld Fld-def split*: *prod.split-asm*)
have *f*: *current* $\Gamma\ (F\ \varepsilon h)$ **if** $\varepsilon h \in$ *Field leq* **for** εh **using** *ε-curr h that*
by(*cases* εh)(*simp add*: *F-simps current-plus-current-minus*)
have *out-ε*: $\varepsilon\ (x, y) = 0$ **if** $(\varepsilon, h) \in$ *Field leq* $x \neq a$ **for** $\varepsilon\ h\ x\ y$
proof(*rule antisym*)
have $\varepsilon\ (x, y) \leq$ *d-OUT* $\varepsilon\ x$ **unfolding** *d-OUT-def* **by**(*rule nn-integral-ge-point*) *simp*
with *OUT-ε*[*OF that*] **show** $\varepsilon\ (x, y) \leq 0$ **by** *simp*
qed *simp*
have *IN-ε*: *d-IN* $\varepsilon\ x = \varepsilon\ (a, x)$ **if** $(\varepsilon, h) \in$ *Field leq* **for** $\varepsilon\ h\ x$
proof(*rule trans*)
show *d-IN* $\varepsilon\ x = (\sum^+ y.\ \varepsilon\ (y, x) *$ *indicator* $\{a\}\ y)$ **unfolding** *d-IN-def*
by(*rule nn-integral-cong*)(*simp add*: *out-ε*[*OF that*] *split*: *split-indicator*)
qed(*simp add*: *max-def ε-curr*[*OF that*])
have *leqI*: $((\varepsilon, h), (\varepsilon', h')) \in$ *leq* **if** $\varepsilon \leq \varepsilon'\ h \leq h'\ (\varepsilon, h) \in$ *Field leq* $(\varepsilon', h') \in$ *Field leq* **for** $\varepsilon\ h\ \varepsilon'\ h'$
using *that* **unfolding** *Fld* **by**(*simp add*: *leq-def in-restrict-rel-iff*)

have *chain-Field*: *Sup M* $\in$ *Field leq* **if** *M*: $M \in$ *Chains leq* **and** *nempty*: $M \neq \{\}$ **for** *M*
unfolding *Sup-prod-def*
proof
from *nempty* **obtain** $\varepsilon\ h$ **where** *in-M*: $(\varepsilon, h) \in M$ **by** *auto*
with *M* **have** *Field*: $(\varepsilon, h) \in$ *Field leq* **by**(*rule Chains-FieldD*)

from *M* **have** *chain*: *Complete-Partial-Order.chain* $(\lambda\varepsilon\ \varepsilon'.\ (\varepsilon, \varepsilon') \in leq)\ M$
by(*intro Chains-into-chain*) *simp*
hence *chain'*: *Complete-Partial-Order.chain* $(\leq)\ M$
by(*auto simp add*: *chain-def leq-def in-restrict-rel-iff*)
hence *chain1*: *Complete-Partial-Order.chain* $(\leq)$ (*fst* ‘ *M*)
and *chain2*: *Complete-Partial-Order.chain* $(\leq)$ (*snd* ‘ *M*)
by(*rule chain-imageI*; *auto*)+

have *outside1*: *Sup* (*fst* ‘ *M*) $(x, y) = 0$ **if** $\neg$ *edge* $\Gamma\ x\ y$ **for** $x\ y$ **using** *that*
by(*auto intro*!: *SUP-eq-const simp add*: *nempty dest*!: *Chains-FieldD*[*OF M*] *ε-curr currentD-outside*)
then have *support-flow* (*Sup* (*fst* ‘ *M*)) $\subseteq$ **E** **by**(*auto elim*!: *support-flow.cases intro*: *ccontr*)
hence *supp-flow1*: *countable* (*support-flow* (*Sup* (*fst* ‘ *M*))) **by**(*rule countable-subset*) *simp*

show *SM1*: *current* Γ (*Sup* (*fst* ‘ *M*))

by(*rule current-Sup*[*OF chain1 - - supp-flow1*])(*auto dest*: *Chains-FieldD*[*OF M*, *THEN ε-curr*] *simp add*: *nempty*)
show *OUT1-na*: *d-OUT* (*Sup* (*fst ‘ M*)) $x = 0$ **if** $x \neq a$ **for** x **using** *that*
by(*subst d-OUT-Sup*[*OF chain1 - supp-flow1*])(*auto simp add*: *nempty intro*!: *SUP-eq-const dest*: *Chains-FieldD*[*OF M*, *THEN OUT-ε*])

interpret *SM1*: *countable-bipartite-web* Γ ⊖ *Sup* (*fst ‘ M*)
using *SM1* **by**(*rule countable-bipartite-web-minus-web*)

let *?h* = *Sup* (*snd ‘ M*)
have *outside2*: *?h* (*x*, *y*) = *0* **if** ¬ *edge* Γ *x y* **for** *x y* **using** *that*
by(*auto intro*!: *SUP-eq-const simp add*: *nempty dest*!: *Chains-FieldD*[*OF M*] *h currentD-outside*)
then have *support-flow ?h* ⊆ **E** **by**(*auto elim*!: *support-flow.cases intro*: *ccontr*)
hence *supp-flow2*: *countable* (*support-flow ?h*) **by**(*rule countable-subset*) *simp*

have *OUT1*: *d-OUT* (*Sup* (*fst ‘ M*)) *x* = (*SUP* (*ε*, *h*)∈*M*. *d-OUT ε x*) **for** *x*
by (*subst d-OUT-Sup* [*OF chain1 - supp-flow1*])
(*simp-all add*: *nempty split-beta image-comp*)
have *OUT1′*: *d-OUT* (*Sup* (*fst ‘ M*)) *x* = (*if x* = *a then SUP* (*ε*, *h*)∈*M*. *d-OUT ε a else 0*) **for** *x*
unfolding *OUT1* **by**(*auto intro*!: *SUP-eq-const simp add*: *nempty OUT-ε dest*!: *Chains-FieldD*[*OF M*])
have *OUT1-le*: ($\bigsqcup$*εh*∈*M*. *d-OUT* (*fst εh*) *x*) ≤ *weight* Γ *x* **for** *x*
using *currentD-weight-OUT*[*OF SM1*, *of x*] *OUT1*[*of x*] **by**(*simp add*: *split-beta*)
have *OUT1-nonneg*: *0* ≤ ($\bigsqcup$*εh*∈*M*. *d-OUT* (*fst εh*) *x*) **for** *x* **using** *in-M* **by**(*rule SUP-upper2*)*simp*
have *IN1*: *d-IN* (*Sup* (*fst ‘ M*)) *x* = (*SUP* (*ε*, *h*)∈*M*. *d-IN ε x*) **for** *x*
by (*subst d-IN-Sup* [*OF chain1 - supp-flow1*])
(*simp-all add*: *nempty split-beta image-comp*)
have *IN1-le*: ($\bigsqcup$*εh*∈*M*. *d-IN* (*fst εh*) *x*) ≤ *weight* Γ *x* **for** *x*
using *currentD-weight-IN*[*OF SM1*, *of x*] *IN1*[*of x*] **by**(*simp add*: *split-beta*)
have *IN1-nonneg*: *0* ≤ ($\bigsqcup$*εh*∈*M*. *d-IN* (*fst εh*) *x*) **for** *x* **using** *in-M* **by**(*rule SUP-upper2*) *simp*
have *IN1′*: *d-IN* (*Sup* (*fst ‘ M*)) *x* = (*SUP* (*ε*, *h*)∈*M*. *ε* (*a*, *x*)) **for** *x*
unfolding *IN1* **by**(*rule SUP-cong*[*OF refl*])(*auto dest*!: *Chains-FieldD*[*OF M*] *IN-ε*)

have *directed*: ∃*εk″*∈*M*. *F* (*snd εk*) + *F* (*fst εk′*) ≤ *F* (*snd εk″*) + *F* (*fst εk″*)
if *mono*: $\bigwedge$*f g*. ($\bigwedge$*z*. *f z* ≤ *g z*) ⟹ *F f* ≤ *F g εk* ∈ *M εk′* ∈ *M*
for *εk εk′* **and** *F* :: - ⇒ *ennreal*
using *chainD*[*OF chain that*(*2*−*3*)]
proof *cases*
case *left*
hence *snd εk* ≤ *snd εk′* **by**(*simp add*: *leq-def less-eq-prod-def in-restrict-rel-iff*)
hence *F* (*snd εk*) + *F* (*fst εk′*) ≤ *F* (*snd εk′*) + *F* (*fst εk′*)
by(*intro add-right-mono mono*)(*clarsimp simp add*: *le-fun-def*)
with *that* **show** *?thesis* **by** *blast*

next
case *right*
hence *fst* $\varepsilon k' \leq$ *fst* εk **by**(*simp add*: *leq-def less-eq-prod-def in-restrict-rel-iff*)
hence F (*snd* εk) + F (*fst* $\varepsilon k'$) $\leq F$ (*snd* εk) + F (*fst* εk)
by(*intro add-left-mono mono*)(*clarsimp simp add*: *le-fun-def*)
with *that* **show** *?thesis* **by** *blast*
qed
have *directed-OUT*: $\exists \varepsilon k''\in M$. *d-OUT* (*snd* εk) x + *d-OUT* (*fst* $\varepsilon k'$) $x \leq$ *d-OUT* (*snd* $\varepsilon k''$) x + *d-OUT* (*fst* $\varepsilon k''$) x
if $\varepsilon k \in M$ $\varepsilon k' \in M$ **for** x εk $\varepsilon k'$ **by**(*rule directed*; *rule d-OUT-mono that*)
have *directed-IN*: $\exists \varepsilon k''\in M$. *d-IN* (*snd* εk) x + *d-IN* (*fst* $\varepsilon k'$) $x \leq$ *d-IN* (*snd* $\varepsilon k''$) x + *d-IN* (*fst* $\varepsilon k''$) x
if $\varepsilon k \in M$ $\varepsilon k' \in M$ **for** x εk $\varepsilon k'$ **by**(*rule directed*; *rule d-IN-mono that*)

let $?\Gamma = \Gamma \ominus$ *Sup* (*fst* ‘ M)

have *hM2*: *current* $?\Gamma$ h **if** εh: $(\varepsilon, h) \in M$ **for** ε h
proof
from εh **have** *Field*: $(\varepsilon, h) \in$ *Field leq* **by**(*rule Chains-FieldD*[*OF M*])
then have *H*: *current* $(\Gamma \ominus \varepsilon)$ h **and** ε*-curr'*: *current* Γ ε **by**(*rule h* ε*-curr*)+
from ε*-curr'* **interpret** Γ: *countable-bipartite-web* $\Gamma \ominus \varepsilon$ **by**(*rule countable-bipartite-web-minus-web*)

fix x
have *d-OUT h x* $\leq$ *d-OUT ?h x* **using** εh **by**(*intro d-OUT-mono*)(*auto intro*: *SUP-upper2*)
also have *OUT*: ... = (*SUP* $h\in$*snd* ‘ M. *d-OUT h x*) **using** *chain2 - supp-flow2*
by(*rule d-OUT-Sup*)(*simp-all add*: *nempty*)
also have ... = ... + (*SUP* $\varepsilon\in$*fst* ‘ M. *d-OUT* ε x) − (*SUP* $\varepsilon\in$*fst* ‘ M. *d-OUT* ε x)
using *OUT1-le*[*of x*]
by (*intro ennreal-add-diff-cancel-right*[*symmetric*] *neq-top-trans*[*OF weight-finite*, *of - x*])
(*simp add*: *image-comp*)
also have ... = (*SUP* $(\varepsilon, k)\in M$. *d-OUT k x* + *d-OUT* ε x) − (*SUP* $\varepsilon\in$*fst* ‘ M. *d-OUT* ε x) **unfolding** *split-def*
by (*subst SUP-add-directed-ennreal*[*OF directed-OUT*])
(*simp-all add*: *image-comp*)
also have (*SUP* $(\varepsilon, k)\in M$. *d-OUT k x* + *d-OUT* ε x) $\leq$ *weight* Γ x
apply(*clarsimp dest*!: *Chains-FieldD*[*OF M*] *intro*!: *SUP-least*)
subgoal premises *that* **for** ε h
using *currentD-weight-OUT*[*OF h*[*OF that*], *of x*] *currentD-weight-OUT*[*OF* ε*-curr*[*OF that*], *of x*]
countable-bipartite-web-minus-web[*OF* ε*-curr*, *THEN countable-bipartite-web.currentD-OUT'*, *OF that h*[*OF that*], **where** *x=x*]
by (*auto simp add*: *ennreal-le-minus-iff split*: *if-split-asm*)
done
also have (*SUP* $\varepsilon\in$*fst* ‘ M. *d-OUT* ε x) = *d-OUT* (*Sup* (*fst* ‘ M)) x **using**

OUT1
by (*simp add*: *split-beta image-comp*)
finally show *d-OUT h x* $\leq$ *weight ?*Γ *x*
using Γ.*currentD-OUT'*[*OF h*[*OF Field*], *of x*] *currentD-weight-IN*[*OF SM1*, *of x*] **by**(*auto simp add*: *ennreal-minus-mono*)

have *d-IN h x* $\leq$ *d-IN ?h x* **using** ε*h* **by**(*intro d-IN-mono*)(*auto intro*: *SUP-upper2*)
also have *IN*: ... = (*SUP h*$\in$*snd ` M*. *d-IN h x*) **using** *chain2 - supp-flow2*
by(*rule d-IN-Sup*)(*simp-all add*: *nempty*)
also have ... = ... + (*SUP* $\varepsilon\in$*fst ` M*. *d-IN* $\varepsilon$ *x*) − (*SUP* $\varepsilon\in$*fst ` M*. *d-IN* ε *x*)
using *IN1-le*[*of x*]
by (*intro ennreal-add-diff-cancel-right* [*symmetric*] *neq-top-trans* [*OF weight-finite*, *of - x*])
(*simp add*: *image-comp*)
also have ... = (*SUP* (ε, *k*)$\in$*M*. *d-IN k x* + *d-IN* ε *x*) − (*SUP* $\varepsilon\in$*fst ` M*. *d-IN* ε *x*) **unfolding** *split-def*
by (*subst SUP-add-directed-ennreal* [*OF directed-IN*])
(*simp-all add*: *image-comp*)
also have (*SUP* (ε, *k*)$\in$*M*. *d-IN k x* + *d-IN* ε *x*) $\leq$ *weight* Γ *x*
apply(*clarsimp dest*!: *Chains-FieldD*[*OF M*] *intro*!: *SUP-least*)
subgoal premises *that* **for** ε *h*
using *currentD-weight-OUT*[*OF h*, *OF that*, **where** *x*=*x*] *currentD-weight-IN*[*OF h*, *OF that*, **where** *x*=*x*]
countable-bipartite-web-minus-web[*OF* ε*-curr*, *THEN countable-bipartite-web.currentD-OUT'*, *OF that h*[*OF that*], **where** *x*=*x*]
currentD-OUT'[*OF* ε*-curr*, *OF that*, **where** *x*=*x*] *currentD-IN*[*OF* ε*-curr*, *OF that*, *of x*] *currentD-weight-IN*[*OF* ε*-curr*, *OF that*, **where** *x*=*x*]
by (*auto simp add*: *ennreal-le-minus-iff image-comp*
split: *if-split-asm intro*: *add-increasing2 order-trans* [*rotated*])
done
also have (*SUP* $\varepsilon\in$*fst ` M*. *d-IN* $\varepsilon$ *x*) = *d-IN* (*Sup* (*fst ` M*)) *x*
using *IN1* **by** (*simp add*: *split-beta image-comp*)
finally show *d-IN h x* $\leq$ *weight ?*Γ *x*
using *currentD-IN*[*OF h*[*OF Field*], *of x*] *currentD-weight-OUT*[*OF SM1*, *of x*]
by(*auto simp add*: *ennreal-minus-mono*)
(*auto simp add*: *ennreal-le-minus-iff add-increasing2*)
show *h e* = *0* **if** $e \notin \mathbf{E}_{?\Gamma}$ **for** *e* **using** *currentD-outside'*[*OF H*, *of e*] *that* **by** *simp*
qed

from *nempty* **have** *snd ` M* $\neq$ {} **by** *simp*
from *chain2 this - supp-flow2* **show** *current*: *current ?*Γ *?h*
by(*rule current-Sup*)(*clarify*; *rule hM2*; *simp*)

have *wM*: *wave ?*Γ *h* **if** (ε, *h*) $\in$ *M* **for** ε *h*
proof

let $?\Gamma' = \Gamma \ominus \varepsilon$
have *subset*: $TER_{?\Gamma'}$ $h \subseteq TER_{?\Gamma}$ h
using *currentD-OUT'*[*OF SM1*] *currentD-OUT'*[*OF* ε*-curr*[*OF Chains-FieldD*[*OF M that*]]] *that*
by(*auto 4 7 elim*!: *SAT.cases intro*: *SAT.intros elim*!: *order-trans*[*rotated*] *intro*: *ennreal-minus-mono d-IN-mono intro*!: *SUP-upper2 split*: *if-split-asm*)

from *h-w*[*OF Chains-FieldD*[*OF M*], *OF that*] **have** *separating* $?\Gamma'$ ($TER_{?\Gamma'}$ h) **by**(*rule waveD-separating*)
then show *separating* $?\Gamma$ ($TER_{?\Gamma}$ h) **using** *subset* **by**(*auto intro*: *separating-weakening*)
qed(*rule hM2*[*OF that*])
show *wave*: *wave* $?\Gamma$ *?h* **using** *chain2* ‹*snd* ‘ $M \neq \{\}$› - *supp-flow2*
by(*rule wave-lub*)(*clarify*; *rule wM*; *simp*)

define *f* **where** $f = F$ (*Sup* (*fst* ‘ *M*), *Sup* (*snd* ‘ *M*))
have *supp-flow*: *countable* (*support-flow f*)
using *supp-flow1 supp-flow2 support-flow-plus-current*[*of Sup* (*fst* ‘ *M*) *?h*]
unfolding *f-def F-simps* **by**(*blast intro*: *countable-subset*)
have *f-alt*: $f =$ *Sup* (($\lambda(\varepsilon, h)$. *plus-current* ε *h*) ‘ *M*)
apply (*simp add*: *fun-eq-iff split-def f-def nempty F-def image-comp*)
apply (*subst* (*1 2*) *add.commute*)
apply (*subst SUP-add-directed-ennreal*)
apply (*rule directed*)
apply (*auto dest*!: *Chains-FieldD* [*OF M*])
done
have *f-curr*: *current* Γ *f* **unfolding** *f-def F-simps* **using** *SM1 current* **by**(*rule current-plus-current-minus*)
have *IN-f*: *d-IN f x* = *d-IN* (*Sup* (*fst* ‘ *M*)) *x* + *d-IN* (*Sup* (*snd* ‘ *M*)) *x* **for** *x*
unfolding *f-def F-simps plus-current-def*
by(*rule d-IN-add SM1 current*)+
have *OUT-f*: *d-OUT f x* = *d-OUT* (*Sup* (*fst* ‘ *M*)) *x* + *d-OUT* (*Sup* (*snd* ‘ *M*)) *x* **for** *x*
unfolding *f-def F-simps plus-current-def*
by(*rule d-OUT-add SM1 current*)+

show $\neg$ *hindered* ($\Gamma \ominus f$) (**is** $\neg$ *hindered* $?\Omega$) — Assertion 6.11
proof
assume *hindered*: *hindered* $?\Omega$
then obtain *g* **where** *g*: *current* $?\Omega$ *g* **and** *g-w*: *wave* $?\Omega$ *g* **and** *hindrance*: *hindrance* $?\Omega$ *g* **by** *cases*
from *hindrance* **obtain** *z* **where** *z*: $z \in A\ \Gamma$ **and** $z\mathcal{E}$: $z \notin \mathcal{E}_{?\Omega}$ ($TER_{?\Omega}$ *g*)
and *OUT-z*: *d-OUT g z* < *weight* $?\Omega$ *z* **by** *cases auto*
define δ **where** $\delta =$ *weight* $?\Omega$ *z* − *d-OUT g z*
have δ*-pos*: $\delta > 0$ **using** *OUT-z* **by**(*simp add*: δ*-def diff-gr0-ennreal del*: *minus-web-sel*)
then have δ*-finite*[*simp*]: $\delta \neq \top$ **using** *z*
by(*simp-all add*: δ*-def*)

have $\exists\,(\varepsilon,\ h) \in M.\ d\text{-}OUT\ f\ a < d\text{-}OUT\ (plus\text{-}current\ \varepsilon\ h)\ a + \delta$
proof(*rule ccontr*)
assume $\neg$ *?thesis*
hence *greater*: $d\text{-}OUT\ (plus\text{-}current\ \varepsilon\ h)\ a + \delta \leq d\text{-}OUT\ f\ a$ **if** $(\varepsilon,\ h) \in M$ **for** ε h **using** *that* **by** *auto*

have *chain''*: *Complete-Partial-Order.chain* $(\leq)$ $((\lambda(\varepsilon,\ h).\ plus\text{-}current\ \varepsilon\ h)$ ‘ $M)$
using *chain'* **by**(*rule chain-imageI*)(*auto simp add*: *le-fun-def add-mono*)

have $d\text{-}OUT\ f\ a + 0 < d\text{-}OUT\ f\ a + \delta$
using *currentD-finite-OUT*[*OF f-curr*, *of a*] **by** (*simp add*: *δ-pos*)
also have $d\text{-}OUT\ f\ a + \delta = (SUP\ (\varepsilon,\ h){\in}M.\ d\text{-}OUT\ (plus\text{-}current\ \varepsilon\ h)\ a) + \delta$
using *chain'' nempty supp-flow*
unfolding *f-alt* **by** (*subst d-OUT-Sup*)
(*simp-all add*: *plus-current-def* [*abs-def*] *split-def image-comp*)
also have $\ldots \leq d\text{-}OUT\ f\ a$
unfolding *ennreal-SUP-add-left*[*symmetric*, *OF nempty*]
proof(*rule SUP-least*, *clarify*)
show $d\text{-}OUT\ (plus\text{-}current\ \varepsilon\ h)\ a + \delta \leq d\text{-}OUT\ f\ a$ **if** $(\varepsilon,\ h) \in M$ **for** ε h
using *greater*[*OF that*] *currentD-finite-OUT*[*OF Chains-FieldD*[*OF M that*, *THEN f*], *of a*]
by(*auto simp add*: *ennreal-le-minus-iff add.commute F-def*)
qed
finally show *False* **by** *simp*
qed
then obtain ε h **where** *hM*: $(\varepsilon,\ h) \in M$ **and** *close*: $d\text{-}OUT\ f\ a < d\text{-}OUT\ (plus\text{-}current\ \varepsilon\ h)\ a + \delta$ **by** *blast*
have *Field*: $(\varepsilon,\ h) \in Field\ leq$ **using** *hM* **by**(*rule Chains-FieldD*[*OF M*])
then have ε: *current* Γ ε
and *unhindered-h*: $\neg$ *hindered* $(\Gamma \ominus F\ (\varepsilon,\ h))$
and *h-curr*: *current* $(\Gamma \ominus \varepsilon)$ h
and *h-w*: *wave* $(\Gamma \ominus \varepsilon)$ h
and *OUT-ε*: $x \neq a \Longrightarrow d\text{-}OUT\ \varepsilon\ x = 0$ **for** x
by(*rule ε-curr OUT-ε h h-w unhindered'*)+
let $?\varepsilon h = plus\text{-}current\ \varepsilon\ h$
have *εh-curr*: *current* Γ $?\varepsilon h$ **using** *Field* **unfolding** *F-simps*[*symmetric*] **by**(*rule f*)

interpret *h*: *countable-bipartite-web* $\Gamma \ominus \varepsilon$ **using** ε **by**(*rule countable-bipartite-web-minus-web*)
interpret *εh*: *countable-bipartite-web* $\Gamma \ominus\ ?\varepsilon h$ **using** *εh-curr* **by**(*rule countable-bipartite-web-minus-web*)
note [*simp del*] = *minus-web-sel*(*2*)
and [*simp*] = *weight-minus-web*[*OF εh-curr*] *weight-minus-web*[*OF SM1*, *simplified*]

define k **where** $k\ e = Sup\ (fst\ ‘\ M)\ e - \varepsilon\ e$ **for** e
have *k-simps*: $k\ (x,\ y) = Sup\ (fst\ ‘\ M)\ (x,\ y) - \varepsilon\ (x,\ y)$ **for** x y **by**(*simp add*:

k-def)
have *k-alt*: *k* (*x*, *y*) = (*if* *x* = *a* ∧ *edge* Γ *x* *y* *then* *Sup* (*fst* ' *M*) (*a*, *y*) − ε (*a*, *y*) *else* *0*) **for** *x* *y*
by (*cases* *x* = *a*)
(*auto simp add*: *k-simps out-*ε [*OF Field*] *currentD-outside* [*OF* ε] *image-comp*
intro!: *SUP-eq-const* [*OF nempty*] *dest*!: *Chains-FieldD* [*OF M*]
intro: *currentD-outside* [*OF* ε*-curr*] *out-*ε)
have *OUT-k*: *d-OUT k x* = (*if* *x* = *a* *then* *d-OUT* (*Sup* (*fst* ' *M*)) *a* − *d-OUT* ε *a* *else* *0*) **for** *x*
proof −
have *d-OUT k x* = (*if* *x* = *a* *then* ($\sum^+$ *y*. *Sup* (*fst* ' *M*) (*a*, *y*) − ε (*a*, *y*)) *else* *0*)
using *currentD-outside*[*OF SM1*] *currentD-outside*[*OF* ε]
by(*auto simp add*: *k-alt d-OUT-def intro*!: *nn-integral-cong*)
also have ($\sum^+$ *y*. *Sup* (*fst* ' *M*) (*a*, *y*) − ε (*a*, *y*)) = *d-OUT* (*Sup* (*fst* '*M*)) *a* − *d-OUT* ε *a*
using *currentD-finite-OUT*[*OF* ε, *of a*] *hM* **unfolding** *d-OUT-def*
by(*subst nn-integral-diff*[*symmetric*])(*auto simp add*: *AE-count-space intro*!: *SUP-upper2*)
finally show *?thesis* **.**
qed
have *IN-k*: *d-IN k y* = (*if* *edge* Γ *a* *y* *then* *Sup* (*fst* ' *M*) (*a*, *y*) − ε (*a*, *y*) *else* *0*) **for** *y*
proof −
have *d-IN k y* = ($\sum^+$ *x*. (*if* *edge* Γ *x* *y* *then* *Sup* (*fst* ' *M*) (*a*, *y*) − ε (*a*, *y*) *else* *0*) ∗ *indicator* {*a*} *x*)
unfolding *d-IN-def* **by**(*rule nn-integral-cong*)(*auto simp add*: *k-alt outgoing-def split*: *split-indicator*)
also have . . . = (*if* *edge* Γ *a* *y* *then* *Sup* (*fst* ' *M*) (*a*, *y*) − ε (*a*, *y*) *else* *0*) **using** *hM*
by(*auto simp add*: *max-def intro*!: *SUP-upper2*)
finally show *?thesis* **.**
qed

have *OUT-*ε*h*: *d-OUT ?*ε*h x* = *d-OUT* ε *x* + *d-OUT h x* **for** *x*
unfolding *plus-current-def* **by**(*rule d-OUT-add*)+
have *IN-*ε*h*: *d-IN ?*ε*h x* = *d-IN* ε *x* + *d-IN h x* **for** *x*
unfolding *plus-current-def* **by**(*rule d-IN-add*)+

have *OUT1-le*′: *d-OUT* (*Sup* (*fst*'*M*)) *x* ≤ *weight* Γ *x* **for** *x*
using *OUT1-le*[*of x*] **unfolding** *OUT1* **by** (*simp add*: *split-beta*′)

have *k*: *current* (Γ ⊖ *?*ε*h*) *k*
proof
fix *x*
show *d-OUT k x* ≤ *weight* (Γ ⊖ *?*ε*h*) *x*
using *a OUT1-na*[*of x*] *currentD-weight-OUT*[*OF hM2*[*OF hM*], *of x*] *currentD-weight-IN*[*OF* ε*h-curr*, *of x*]

currentD-weight-IN[*OF* ε, *of* x] *OUT1-le′*[*of* x]
apply(*auto simp add*: *diff-add-eq-diff-diff-swap-ennreal diff-add-assoc2-ennreal*[*symmetric*]
OUT-k OUT-ε *OUT-*ε*h IN-*ε*h currentD-OUT′*[*OF* ε]
IN-ε[*OF Field*] *h.currentD-OUT′*[*OF h-curr*])
apply(*subst diff-diff-commute-ennreal*)
apply(*intro ennreal-minus-mono*)
apply(*auto simp add*: *ennreal-le-minus-iff ac-simps less-imp-le OUT1*)
done

have $*$: ($\bigsqcup$*xa*$\in$*M*. *fst xa* (*a*, *x*)) $\leq$ *d-IN* (*Sup* (*fst*‘*M*)) *x*
unfolding *IN1* **by** (*intro SUP-subset-mono*) (*auto simp*: *split-beta′*
d-IN-ge-point)
also have ... $\leq$ *weight* Γ *x*
using *IN1-le*[*of x*] *IN1* **by** (*simp add*: *split-beta′*)
finally show *d-IN k x* $\leq$ *weight* (Γ $\ominus$ *?*ε*h*) *x*
using *currentD-weight-IN*[*OF* ε*h-curr*, *of x*] *currentD-weight-OUT*[*OF*
ε*h-curr*, *of x*]
currentD-weight-IN[*OF hM2*[*OF hM*], *of x*] *IN-*ε[*OF Field*, *of x*] $*$
apply (*auto simp add*: *IN-k outgoing-def IN-*ε*h IN-*ε *A-in diff-add-eq-diff-diff-swap-ennreal*)
apply (*subst diff-diff-commute-ennreal*)
apply (*intro ennreal-minus-mono*[*OF - order-refl*])
apply (*auto simp add*: *ennreal-le-minus-iff ac-simps image-comp intro*:
order-trans add-mono)
done
show *k e* = *0* **if** $e \notin \mathbf{E}_{\Gamma \ominus ?\varepsilon h}$ **for** *e*
using *that* **by** (*cases e*) (*simp add*: *k-alt*)
qed

define *q* **where** *q* = ($\sum^+$ *y*$\in$*B* (Γ $\ominus$ *?*ε*h*). *d-IN k y* $-$ *d-OUT k y*)
have *q-alt*: *q* = ($\sum^+$ *y*$\in-$ *A* (Γ $\ominus$ *?*ε*h*). *d-IN k y* $-$ *d-OUT k y*) **using**
disjoint
by(*auto simp add*: *q-def nn-integral-count-space-indicator currentD-outside-OUT*[*OF*
k] *currentD-outside-IN*[*OF k*] *not-vertex split*: *split-indicator intro*!: *nn-integral-cong*)
have *q-simps*: *q* = *d-OUT* (*Sup* (*fst* ‘ *M*)) *a* $-$ *d-OUT* ε *a*
proof $-$
have *q* = ($\sum^+$ *y*. *d-IN k y*) **using** *a IN1 OUT1 OUT1-na* **unfolding** *q-alt*
by(*auto simp add*: *nn-integral-count-space-indicator OUT-k IN-*ε[*OF Field*]
OUT-ε *currentD-outside*[*OF* ε] *outgoing-def no-loop A-in IN-k intro*!: *nn-integral-cong*
split: *split-indicator*)
also have ... = *d-OUT* (*Sup* (*fst* ‘ *M*)) *a* $-$ *d-OUT* ε *a* **using** *cur-
rentD-finite-OUT*[*OF* ε, *of a*] *hM currentD-outside*[*OF SM1*] *currentD-outside*[*OF*
ε]
by(*subst d-OUT-diff*[*symmetric*])(*auto simp add*: *d-OUT-def IN-k intro*!:
SUP-upper2 nn-integral-cong)
finally show *?thesis* **.**
qed
have *q-finite*: $q \neq \top$ **using** *currentD-finite-OUT*[*OF SM1*, *of a*]
by(*simp add*: *q-simps*)
have *q-nonneg*: $0 \leq q$ **using** *hM* **by**(*auto simp add*: *q-simps intro*!: *d-OUT-mono*

SUP-upper2)
have *q-less-δ*: *q* < *δ* **using** *close*
unfolding *q-simps δ-def OUT-εh OUT-f*
proof −
let *?F* = *d-OUT* (*Sup* (*fst'M*)) *a* **and** *?S* = *d-OUT* (*Sup* (*snd'M*)) *a*
and *?ε* = *d-OUT* *ε* *a* **and** *?h* = *d-OUT* *h* *a* **and** *?w* = *weight* (Γ ⊖ *f*) *z*
− *d-OUT* *g* *z*
have *?F* + *?h* ≤ *?F* + *?S*
using *hM* **by** (*auto intro*!: *add-mono d-OUT-mono SUP-upper2*)
also assume *?F* + *?S* < *?ε* + *?h* + *?w*
finally have *?h* + *?F* < *?h* + (*?w* + *?ε*)
by (*simp add*: *ac-simps*)
then show *?F* − *?ε* < *?w*
using *currentD-finite-OUT*[*OF ε*, *of a*] *hM* **unfolding** *ennreal-add-left-cancel-less*
by (*subst minus-less-iff-ennreal*) (*auto intro*!: *d-OUT-mono SUP-upper2*
simp: *less-top*)
qed

define *g′* **where** *g′* = *plus-current g* (*Sup* (*snd* ' *M*) − *h*)
have *g′-simps*: *g′ e* = *g e* + *Sup* (*snd* ' *M*) *e* − *h e* **for** *e*
using *hM* **by**(*auto simp add*: *g′-def intro*!: *add-diff-eq-ennreal intro*: *SUP-upper2*)
have *OUT-g′*: *d-OUT g′ x* = *d-OUT g x* + (*d-OUT* (*Sup* (*snd* ' *M*)) *x* −
d-OUT h x) **for** *x*
unfolding *g′-simps*[*abs-def*] **using** *εh.currentD-finite-OUT*[*OF k*] *hM*
h.currentD-finite-OUT[*OF h-curr*] *hM*
apply(*subst d-OUT-diff*)
apply(*auto simp add*: *add-diff-eq-ennreal*[*symmetric*] *k-simps intro*: *add-increasing*
intro!: *SUP-upper2*)
apply(*subst d-OUT-add*)
apply(*auto simp add*: *add-diff-eq-ennreal*[*symmetric*] *k-simps intro*: *add-increasing*
intro!:)
apply(*simp add*: *add-diff-eq-ennreal SUP-apply*[*abs-def*])
apply(*auto simp add*: *g′-def image-comp intro*!: *add-diff-eq-ennreal*[*symmetric*]
d-OUT-mono intro: *SUP-upper2*)
done
have *IN-g′*: *d-IN g′ x* = *d-IN g x* + (*d-IN* (*Sup* (*snd* ' *M*)) *x* − *d-IN h x*) **for**
x
unfolding *g′-simps*[*abs-def*] **using** *εh.currentD-finite-IN*[*OF k*] *hM h.currentD-finite-IN*[*OF*
h-curr] *hM*
apply(*subst d-IN-diff*)
apply(*auto simp add*: *add-diff-eq-ennreal*[*symmetric*] *k-simps intro*: *add-increasing*
intro!: *SUP-upper2*)
apply(*subst d-IN-add*)
apply(*auto simp add*: *add-diff-eq-ennreal*[*symmetric*] *k-simps intro*: *add-increasing*
intro!: *SUP-upper*)
apply(*auto simp add*: *g′-def SUP-apply*[*abs-def*] *image-comp intro*!: *add-diff-eq-ennreal*[*symmetric*]
d-IN-mono intro: *SUP-upper2*)
done

have h': *current* ($\Gamma \ominus$ *Sup* (*fst* ‘ M)) h **using** *hM* **by**(*rule hM2*)

let *?*$\Gamma = \Gamma \ominus$ *?*$\varepsilon h \ominus k$
interpret Γ: *web ?*Γ **using** k **by**(*rule* εh*.web-minus-web*)
note [*simp*] = εh*.weight-minus-web*[*OF k*] *h.weight-minus-web*[*OF h-curr*] *weight-minus-web*[*OF f-curr*] *SM1.weight-minus-web*[*OF* h', *simplified*]

interpret Ω: *countable-bipartite-web* $\Gamma \ominus f$ **using** *f-curr* **by**(*rule countable-bipartite-web-minus-web*)

have $*$: $\Gamma \ominus f = \Gamma \ominus$ *Sup* (*fst* ‘ M) $\ominus$ *Sup* (*snd* ‘ M) **unfolding** *f-def F-simps* **using** *SM1 current* **by**(*rule minus-plus-current*)
have *OUT-*εk: *d-OUT* (*Sup* (*fst* ‘ M)) x = *d-OUT* ε x + *d-OUT* k x **for** x
using *OUT1'*[*of x*] *currentD-finite-OUT*[*OF* ε] *hM*
by(*auto simp add*: *OUT-k OUT-*ε *add-diff-self-ennreal SUP-upper2*)
have *IN-*εk: *d-IN* (*Sup* (*fst* ‘ M)) x = *d-IN* ε x + *d-IN* k x **for** x
using *IN1'*[*of x*] *currentD-finite-IN*[*OF* ε] *currentD-outside*[*OF* ε] *currentD-outside*[*OF* ε*-curr*]
by(*auto simp add*: *IN-k IN-*ε[*OF Field*] *add-diff-self-ennreal split-beta nempty image-comp*
dest!: *Chains-FieldD*[*OF M*] *intro*!: *SUP-eq-const intro*: *SUP-upper2*[*OF hM*])
have $**$: *?*$\Gamma = \Gamma \ominus$ *Sup* (*fst* ‘ M) $\ominus h$
proof(*rule web.equality*)
show *weight ?*Γ = *weight* ($\Gamma \ominus$ *Sup* (*fst* ‘ M) $\ominus h$)
using *OUT-*εk *OUT-*εh *currentD-finite-OUT*[*OF* ε] *IN-*εk *IN-*εh *currentD-finite-IN*[*OF* ε]
by(*auto simp add*: *diff-add-eq-diff-diff-swap-ennreal diff-diff-commute-ennreal*)
qed *simp-all*
have g'*-alt*: g' = *plus-current* (*Sup* (*snd* ‘ M)) $g - h$
by(*simp add*: *fun-eq-iff* g'*-simps add-diff-eq-ennreal add.commute*)

have *current* ($\Gamma \ominus$ *Sup* (*fst* ‘ M)) (*plus-current* (*Sup* (*snd* ‘ M)) g) **using** *current g* **unfolding** $*$
by(*rule SM1.current-plus-current-minus*)
hence g': *current ?*Γ g' **unfolding** $*$ $**$ g'*-alt* **using** *hM2*[*OF hM*]
by(*rule SM1.current-minus*)(*auto intro*!: *add-increasing2 SUP-upper2 hM*)

have *wave* ($\Gamma \ominus$ *Sup* (*fst* ‘ M)) (*plus-current* (*Sup* (*snd* ‘ M)) g) **using** *current wave g g-w*
unfolding $*$ **by**(*rule SM1.wave-plus-current-minus*)
then have g'*-w*: *wave ?*Γ g' **unfolding** $*$ $**$ g'*-alt* **using** *hM2*[*OF hM*]
by(*rule SM1.wave-minus*)(*auto intro*!: *add-increasing2 SUP-upper2 hM*)

have *hindrance-by ?*Γ g' q
proof
show $z \in A$ *?*Γ **using** z **by** *simp*
show $z \notin \mathcal{E}_{?\Gamma}$ (*TER*$_{?\Gamma}$ g')
proof

assume $z \in \mathcal{E}_{?\Gamma}$ $(TER_{?\Gamma}\ g')$
hence $OUT\text{-}z$: $d\text{-}OUT\ g'\ z = 0$
and *ess*: *essential* $?\Gamma$ $(B\ \Gamma)$ $(TER_{?\Gamma}\ g')$ z **by**(*simp-all add*: *SINK.simps*)
from *ess* **obtain** p y **where** p: *path* Γ z p y **and** y: $y \in B\ \Gamma$
and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF\ (TER_{?\Gamma}\ g')$ **by**(*rule essentialE-RF*) *auto*
from y **have** y': $y \notin A\ \Gamma$ **using** *disjoint* **by** *blast*
from p z y **obtain** *py*: $p = [y]$ **and** *edge*: *edge* Γ z y **using** *disjoint*
by(*cases*)(*auto 4 3 elim*: *rtrancl-path.cases dest*: *bipartite-E*)
hence *yRF*: $y \notin RF\ (TER_{?\Gamma}\ g')$ **using** *bypass*[*of y*] **by**(*auto*)
with *wave-not-RF-IN-zero*[*OF g' g'-w*, *of y*] **have** $IN\text{-}g'\text{-}y$: $d\text{-}IN\ g'\ y = 0$
by(*auto intro*: *roofed-greaterI*)
with *yRF* y y' **have** *w-y*: *weight* $?\Gamma$ $y > 0$ **using** *currentD-OUT*[*OF g'*, *of y*]
by(*auto simp add*: *RF-in-B currentD-SAT*[*OF g'*] *SINK.simps zero-less-iff-neq-zero*)
have $y \notin SAT\ (\Gamma \ominus f)\ g$
proof
assume $y \in SAT\ (\Gamma \ominus f)\ g$
with y *disjoint* **have** *IN-g-y*: $d\text{-}IN\ g\ y = weight\ (\Gamma \ominus f)\ y$ **by**(*auto simp add*: *currentD-SAT*[*OF g*])
have $0 < weight\ \Gamma\ y - d\text{-}IN\ (\bigsqcup x{\in}M.\ fst\ x)\ y - d\text{-}IN\ h\ y$
using y' *w-y* **unfolding** $**$ **by** *auto*
have $d\text{-}IN\ g'\ y > 0$
using y' *w-y hM* **unfolding** $**$
apply(*simp add*: *IN-g' IN-f IN-g-y diff-add-eq-diff-diff-swap-ennreal*)
apply(*subst add-diff-eq-ennreal*)
apply(*auto intro*!: *SUP-upper2 d-IN-mono simp*: *diff-add-self-ennreal diff-gt-0-iff-gt-ennreal*)
done
with $IN\text{-}g'\text{-}y$ **show** *False* **by** *simp*
qed
then have $y \notin TER_{\Gamma \ominus f}\ g$ **by** *simp*
with p y *py* **have** *essential* Γ $(B\ \Gamma)$ $(TER_{\Gamma \ominus f}\ g)$ z **by**(*auto intro*: *essentialI*)
moreover with z *waveD-separating*[*OF g-w*, *THEN separating-RF-A*] **have** $z \in \mathcal{E}_{?\Omega}$ $(TER_{?\Omega}\ g)$
by(*auto simp add*: *RF-in-essential*)
with $z\mathcal{E}$ **show** *False* **by** *contradiction*
qed
have $\delta \leq weight\ ?\Gamma\ z - d\text{-}OUT\ g'\ z$
unfolding $**$ $OUT\text{-}g'$ **using** z
apply (*simp add*: *δ-def OUT-f diff-add-eq-diff-diff-swap-ennreal*)
apply (*subst* (*5*) *diff-diff-commute-ennreal*)
apply (*rule ennreal-minus-mono*[*OF - order-refl*])
apply (*auto simp add*: *ac-simps diff-add-eq-diff-diff-swap-ennreal*[*symmetric*] *add-diff-self-ennreal image-comp*
intro!: *ennreal-minus-mono*[*OF order-refl*] *SUP-upper2*[*OF hM*] *d-OUT-mono*)
done

then show *q-z*: $q < weight\ ?\Gamma\ z - d\text{-}OUT\ g'\ z$ **using** *q-less-δ* **by** *simp*
then show $d\text{-}OUT\ g'\ z < weight\ ?\Gamma\ z$ **using** *q-nonneg z*
by(*auto simp add*: *less-diff-eq-ennreal less-top*[*symmetric*] *ac-simps* Γ.*currentD-finite-OUT*[*OF g'*]
intro: *le-less-trans*[*rotated*] *add-increasing*)
qed
then have *hindered-by*: *hindered-by* $(\Gamma \ominus ?\varepsilon h \ominus k)\ q$ **using** g' g'*-w* **by**(*rule hindered-by.intros*)
then have *hindered* $(\Gamma \ominus ?\varepsilon h)$ **using** *q-finite* **unfolding** *q-def* **by** −(*rule εh.hindered-reduce-current*[*OF k*])
with *unhindered-h* **show** *False* **unfolding** *F-simps* **by** *contradiction*
qed
qed

define *sat* **where** *sat* =
(λ(ε, h).
let
$f = F\ (\varepsilon, h)$;
$k = SOME\ k.\ current\ (\Gamma \ominus f)\ k \wedge wave\ (\Gamma \ominus f)\ k \wedge (\forall k'.\ current\ (\Gamma \ominus f)\ k' \wedge wave\ (\Gamma \ominus f)\ k' \wedge k \leq k' \longrightarrow k = k')$
in
if $d\text{-}OUT\ (plus\text{-}current\ f\ k)\ a < weight\ \Gamma\ a$ *then*
let
$\Omega = \Gamma \ominus f \ominus k$;
$y = SOME\ y.\ y \in \mathbf{OUT}_{\Omega}\ a \wedge weight\ \Omega\ y > 0$;
$\delta = SOME\ \delta.\ \delta > 0 \wedge \delta < enn2real\ (min\ (weight\ \Omega\ a)\ (weight\ \Omega\ y)) \wedge \neg\ hindered\ (reduce\text{-}weight\ \Omega\ y\ \delta)$
in
(*plus-current* ε (*zero-current*((*a*, *y*) := δ)), *plus-current h k*)
else (ε, *h*))

have *zero*: (*zero-current*, *zero-current*) ∈ *Field leq*
by(*rule F-I*)(*simp-all add*: *unhindered F-def*)

have *a-TER*: $a \in TER_{\Gamma \ominus F\ \varepsilon h}\ k$
if *that*: εh ∈ *Field leq*
and *k*: *current* (Γ ⊖ *F* εh) *k* **and** *k-w*: *wave* (Γ ⊖ *F* εh) *k*
and *less*: $d\text{-}OUT\ (plus\text{-}current\ (F\ \varepsilon h)\ k)\ a < weight\ \Gamma\ a$ **for** εh *k*
proof(*rule ccontr*)
assume ¬ *?thesis*
hence $\mathcal{E}$: $a \notin \mathcal{E}_{\Gamma \ominus F\ \varepsilon h}\ (TER_{\Gamma \ominus F\ \varepsilon h}\ k)$ **by** *auto*
from *that* **have** *f*: *current* Γ (*F* εh) **and** *unhindered*: ¬ *hindered* (Γ ⊖ *F* εh)
by(*cases* εh; *simp add*: *f unhindered'*; *fail*)+

from *less* **have** $d\text{-}OUT\ k\ a < weight\ (\Gamma \ominus F\ \varepsilon h)\ a$ **using** *a currentD-finite-OUT*[*OF f*, *of a*]
by(*simp add*: *d-OUT-def nn-integral-add less-diff-eq-ennreal add.commute less-top*[*symmetric*])
with - $\mathcal{E}$ **have** *hindrance* (Γ ⊖ *F* εh) *k* **by**(*rule hindrance*)(*simp add*: *a*)

then have *hindered* $(\Gamma \ominus F\ \varepsilon h)$ **using** *k k-w* **..**
with *unhindered* **show** *False* **by** *contradiction*
qed

note *minus-web-sel*(*2*)[*simp del*]

let *?P-y* $= \lambda\varepsilon h\ k\ y.\ y \in \mathbf{OUT}_{\Gamma \ominus F\ \varepsilon h \ominus k}\ a \wedge weight\ (\Gamma \ominus F\ \varepsilon h \ominus k)\ y > 0$
have *Ex-y*: *Ex* (*?P-y* εh *k*)
if *that*: $\varepsilon h \in Field\ leq$
and *k*: *current* $(\Gamma \ominus F\ \varepsilon h)$ *k* **and** *k-w*: *wave* $(\Gamma \ominus F\ \varepsilon h)$ *k*
and *less*: *d-OUT* (*plus-current* $(F\ \varepsilon h)$ *k*) $a <$ *weight* Γ *a* **for** εh *k*
proof(*rule ccontr*)
let $?\Omega = \Gamma \ominus F\ \varepsilon h \ominus k$
assume $*$: $\neg$ *?thesis*

interpret Γ: *countable-bipartite-web* $\Gamma \ominus F\ \varepsilon h$ **using** *f*[*OF that*] **by**(*rule countable-bipartite-web-minus-web*)
note [*simp*] = *weight-minus-web*[*OF f*[*OF that*]] Γ.*weight-minus-web*[*OF k*]

have *hindrance* $?\Omega$ *zero-current*
proof
show $a \in A\ ?\Omega$ **using** *a* **by** *simp*
show $a \notin \mathcal{E}_{?\Omega}$ (*TER*$_{?\Omega}$ *zero-current*) (**is** $a \notin \mathcal{E}_{-}$ *?TER*)
proof
assume $a \in \mathcal{E}_{?\Omega}$ *?TER*
then obtain *p y* **where** *p*: *path* $?\Omega$ *a p y* **and** *y*: $y \in B\ ?\Omega$
and *bypass*: $\bigwedge z.\ z \in set\ p \Longrightarrow z \notin RF_{?\Omega}$ *?TER* **by**(*rule* $\mathcal{E}$*-E-RF*)(*auto*)
from *a p y disjoint* **have** *Nil*: $p \neq []$ **by**(*auto simp add*: *rtrancl-path-simps*)
hence *edge* $?\Omega$ *a* (*p* ! *0*) *p* ! *0* $\notin RF_{?\Omega}$ *?TER*
using *rtrancl-path-nth*[*OF p*, *of 0*] *bypass* **by** *auto*
with $*$ **show** *False* **by**(*auto simp add*: *not-less outgoing-def intro*: *roofed-greaterI*)
qed

have *d-OUT* (*plus-current* $(F\ \varepsilon h)$ *k*) $x =$ *d-OUT* $(F\ \varepsilon h)$ $x +$ *d-OUT k x* **for** *x*
by(*simp add*: *d-OUT-def nn-integral-add*)
then show *d-OUT zero-current a* $<$ *weight* $?\Omega$ *a* **using** *less a-TER*[*OF that k k-w less*] *a*
by(*simp add*: *SINK.simps diff-gr0-ennreal*)
qed
hence *hindered* $?\Omega$
by(*auto intro*!: *hindered.intros order-trans*[*OF currentD-weight-OUT*[*OF k*]] *order-trans*[*OF currentD-weight-IN*[*OF k*]])
moreover have $\neg$ *hindered* $?\Omega$ **using** *unhindered'*[*OF that*] *k k-w* **by**(*rule* Γ.*unhindered-minus-web*)
ultimately show *False* **by** *contradiction*
qed

have *increasing*: $\varepsilon h \leq sat\ \varepsilon h \wedge sat\ \varepsilon h \in Field\ leq$ **if** $\varepsilon h \in Field\ leq$ **for** εh

proof(*cases* εh)
case (*Pair* ε *h*)
with *that* **have** *that*: $(\varepsilon, h) \in$ *Field leq* **by** *simp*
have *f*: *current* Γ (F (ε, h)) **and** *unhindered*: $\neg$ *hindered* ($\Gamma \ominus F$ (ε, h))
and ε: *current* Γ ε
and *h*: *current* $(\Gamma \ominus \varepsilon)$ *h* **and** *h-w*: *wave* $(\Gamma \ominus \varepsilon)$ *h* **and** *OUT-ε*: $x \neq a \Longrightarrow$ *d-OUT* ε $x = 0$ **for** *x*
using *that* **by**(*rule f unhindered′ ε-curr OUT-ε h h-w*)+
interpret Γ: *countable-bipartite-web* $\Gamma \ominus F$ (ε, h) **using** *f* **by**(*rule countable-bipartite-web-minus-web*)
note [*simp*] = *weight-minus-web*[*OF f*]

let *?P-k* = λk. *current* ($\Gamma \ominus F$ (ε, h)) k $\wedge$ *wave* ($\Gamma \ominus F$ (ε, h)) k $\wedge$ ($\forall k'$. *current* ($\Gamma \ominus F$ (ε, h)) k' $\wedge$ *wave* ($\Gamma \ominus F$ (ε, h)) k' $\wedge$ $k \leq k' \longrightarrow k = k'$)
define *k* **where** *k* = *Eps ?P-k*
have *Ex ?P-k* **by**(*intro ex-maximal-wave*)(*simp-all*)
hence *?P-k k* **unfolding** *k-def* **by**(*rule someI-ex*)
hence *k*: *current* ($\Gamma \ominus F$ (ε, h)) *k* **and** *k-w*: *wave* ($\Gamma \ominus F$ (ε, h)) *k*
and *maximal*: $\bigwedge k'$. $[\![$ *current* ($\Gamma \ominus F$ (ε, h)) k'; *wave* ($\Gamma \ominus F$ (ε, h)) k'; $k \leq k'$ $]\!] \Longrightarrow k = k'$ **by** *blast*+
note [*simp*] = Γ.*weight-minus-web*[*OF k*]

let *?fk* = *plus-current* (F (ε, h)) *k*
have *IN-fk*: *d-IN ?fk x* = *d-IN* (F (ε, h)) *x* + *d-IN k x* **for** *x*
by(*simp add*: *d-IN-def nn-integral-add*)
have *OUT-fk*: *d-OUT ?fk x* = *d-OUT* (F (ε, h)) *x* + *d-OUT k x* **for** *x*
by(*simp add*: *d-OUT-def nn-integral-add*)
have *fk*: *current* Γ *?fk* **using** *f k* **by**(*rule current-plus-current-minus*)

show *?thesis*
proof(*cases d-OUT ?fk a* < *weight* Γ *a*)
case *less*: *True*

define Ω **where** $\Omega = \Gamma \ominus F$ $(\varepsilon, h) \ominus k$
have *B-Ω* [*simp*]: *B* Ω = *B* Γ **by**(*simp add*: Ω-*def*)

have *loose*: *loose* Ω **unfolding** Ω-*def* **using** *unhindered k k-w maximal* **by**(*rule* Γ.*loose-minus-web*)
interpret Ω: *countable-bipartite-web* Ω **using** *k* **unfolding** Ω-*def*
by(*rule* Γ.*countable-bipartite-web-minus-web*)

have *a-$\mathcal{E}$*: $a \in TER_{\Gamma \ominus F\ (\varepsilon,\ h)}$ *k* **using** *that k k-w less* **by**(*rule a-TER*)
then have *weight-Ω-a*: *weight* Ω *a* = *weight* Γ *a* − *d-OUT* (F (ε, h)) *a*
using *a disjoint* **by**(*auto simp add*: *roofed-circ-def* Ω-*def SINK.simps*)
then have *weight-a*: *0* < *weight* Ω *a* **using** *less a-$\mathcal{E}$*
by(*simp add*: *OUT-fk SINK.simps diff-gr0-ennreal*)

let *?P-y* = λy. $y \in \mathbf{OUT}_\Omega$ a $\wedge$ *weight* Ω $y > 0$
define *y* **where** *y* = *Eps ?P-y*

have *Ex ?P-y* **using** *that k k-w less* **unfolding** *Ω-def* **by**(*rule Ex-y*)
hence *?P-y y* **unfolding** *y-def* **by**(*rule someI-ex*)
hence *y-OUT*: $y \in \mathbf{OUT}_{\Omega}$ *a* **and** *weight-y*: *weight Ω y > 0* **by** *blast+*
from *y-OUT* **have** *y-B*: $y \in B\ \Omega$ **by**(*auto simp add*: *outgoing-def Ω-def dest*: *bipartite-E*)
with *weight-y* **have** *yRF*: $y \notin RF_{\Gamma \ominus F\ (\varepsilon,\ h)}\ (TER_{\Gamma \ominus F\ (\varepsilon,\ h)}\ k)$
unfolding *Ω-def* **using** *currentD-OUT*[*OF k*, *of y*] *disjoint*
by(*auto split*: *if-split-asm simp add*: *SINK.simps currentD-SAT*[*OF k*] *roofed-circ-def RF-in-B Γ.currentD-finite-IN*[*OF k*])
hence *IN-k-y*: *d-IN k y = 0* **by**(*rule wave-not-RF-IN-zero*[*OF k k-w*])

define *bound* **where** *bound = enn2real (min (weight Ω a) (weight Ω y))*
have *bound-pos*: *bound > 0* **using** *weight-y weight-a* **using** *Ω.weight-finite*
by(*cases weight Ω a weight Ω y rule*: *ennreal2-cases*)
(*simp-all add*: *bound-def min-def split*: *if-split-asm*)

let *?P-δ* = $\lambda\delta.\ \delta > 0 \wedge \delta < bound \wedge \neg$ *hindered* (*reduce-weight Ω y δ*)
define *δ* **where** *δ = Eps ?P-δ*
let *?Ω = reduce-weight Ω y δ*

from *Ω.unhinder*[*OF loose - weight-y bound-pos*] *y-B disjoint*
have *Ex ?P-δ* **by**(*auto simp add*: *Ω-def*)
hence *?P-δ δ* **unfolding** *δ-def* **by**(*rule someI-ex*)
hence *δ-pos*: *0 < δ* **and** *δ-le-bound*: *δ < bound* **and** *unhindered′*: ¬ *hindered ?Ω* **by** *blast+*
from *δ-pos* **have** *δ-nonneg*: $0 \le \delta$ **by** *simp*
from *δ-le-bound δ-pos* **have** *δ-le-a*: *δ < weight Ω a* **and** *δ-le-y*: *δ < weight Ω y*
by(*cases weight Ω a weight Ω y rule*: *ennreal2-cases*;
simp add: *bound-def min-def ennreal-less-iff split*: *if-split-asm*)+

let *?Γ = Γ ⊖ ?fk*
interpret *Γ′*: *countable-bipartite-web ?Γ* **by**(*rule countable-bipartite-web-minus-web fk*)+
note [*simp*] = *weight-minus-web*[*OF fk*]

let *?g = zero-current*((*a*, *y*) := *δ*)
have *OUT-g*: *d-OUT ?g x =* (*if x = a then δ else 0*) **for** *x*
proof(*rule trans*)
show *d-OUT ?g x* = ($\sum^+$ *z*. (*if x = a then δ else 0*) ∗ *indicator* {*y*} *z*) **unfolding** *d-OUT-def*
by(*rule nn-integral-cong*) *simp*
show … = (*if x = a then δ else 0*) **using** *δ-pos* **by**(*simp add*: *max-def*)
qed
have *IN-g*: *d-IN ?g x =* (*if x = y then δ else 0*) **for** *x*
proof(*rule trans*)
show *d-IN ?g x* = ($\sum^+$ *z*. (*if x = y then δ else 0*) ∗ *indicator* {*a*} *z*) **unfolding** *d-IN-def*
by(*rule nn-integral-cong*) *simp*

show ... = (*if* $x = y$ *then* δ *else* 0) **using** *δ-pos* **by**(*simp add*: *max-def*)
qed

have *g*: *current* ?Γ ?*g*
proof
show *d-OUT* ?*g* $x \leq$ *weight* ?Γ x **for** x
proof(*cases* $x = a$)
case *False*
then show ?*thesis* **using** *currentD-weight-OUT*[*OF fk*, *of x*] *currentD-weight-IN*[*OF fk*, *of x*]
by(*auto simp add*: *OUT-g zero-ennreal-def*[*symmetric*])
next
case *True*
then show ?*thesis* **using** *δ-le-a a a-ℰ δ-pos* **unfolding** *OUT-g*
by(*simp add*: *OUT-g Ω-def SINK.simps OUT-fk split*: *if-split-asm*)
qed
show *d-IN* ?*g* $x \leq$ *weight* ?Γ x **for** x
proof(*cases* $x = y$)
case *False*
then show ?*thesis* **using** *currentD-weight-OUT*[*OF fk*, *of x*] *currentD-weight-IN*[*OF fk*, *of x*]
by(*auto simp add*: *IN-g zero-ennreal-def*[*symmetric*])
next
case *True*
then show ?*thesis* **using** *δ-le-y y-B a-ℰ δ-pos currentD-OUT*[*OF k*, *of y*] *IN-k-y*
by(*simp add*: *OUT-g Ω-def SINK.simps OUT-fk IN-fk IN-g split*: *if-split-asm*)
qed
show ?*g* $e = 0$ **if** $e \notin \mathbf{E}_{?\Gamma}$ **for** e **using** *y-OUT that* **by**(*auto simp add*: *Ω-def outgoing-def*)
qed
interpret Γ″: *web* Γ ⊖ ?*fk* ⊖ ?*g* **using** *g* **by**(*rule* Γ′.*web-minus-web*)

let ?ε′ = *plus-current* ε (*zero-current*((*a*, *y*) := δ))
let ?*h*′ = *plus-current h k*
have *F*′: *F* (?ε′, ?*h*′) = *plus-current* (*plus-current* (*F* (ε, *h*)) *k*) (*zero-current*((*a*, *y*) := δ)) (**is** - = ?*f*′)
by(*auto simp add*: *F-simps fun-eq-iff add-ac*)
have *sat*: *sat* (ε, *h*) = (?ε′, ?*h*′) **using** *less*
by(*simp add*: *sat-def k-def Ω-def Let-def y-def bound-def δ-def*)

have *le*: (ε, *h*) ≤ (?ε′, ?*h*′) **using** *δ-pos*
by(*auto simp add*: *le-fun-def add-increasing2 add-increasing*)

have *current* (Γ ⊖ ε) ((λ-. *0*)((*a*, *y*) := *ennreal* δ)) **using** *g*
by(*rule current-weight-mono*)(*auto simp add*: *weight-minus-web*[*OF* ε] *intro*!: *ennreal-minus-mono d-OUT-mono d-IN-mono*, *simp-all add*: *F-def add-increasing2*)
with ε **have** ε′: *current* Γ ?ε′ **by**(*rule current-plus-current-minus*)

moreover have *eq-0*: *d-OUT ?ε′ x = 0* **if** $x \neq a$ **for** *x* **unfolding** *plus-current-def* **using** *that*
by(*subst d-OUT-add*)(*simp-all add*: *δ-nonneg d-OUT-fun-upd OUT-ε*)
moreover
from *ε′* **interpret** *ε′*: *countable-bipartite-web* Γ ⊖ *?ε′* **by**(*rule countable-bipartite-web-minus-web*)
from *ε* **interpret** *ε*: *countable-bipartite-web* Γ ⊖ *ε* **by**(*rule countable-bipartite-web-minus-web*)
have *g′*: *current* (Γ ⊖ *ε*) *?g* **using** *g*
apply(*rule current-weight-mono*)
apply(*auto simp add*: *weight-minus-web*[*OF ε*] *intro*!: *ennreal-minus-mono d-OUT-mono d-IN-mono*)
apply(*simp-all add*: *F-def add-increasing2*)
done
have *k′*: *current* (Γ ⊖ *ε* ⊖ *h*) *k* **using** *k* **unfolding** *F-simps minus-plus-current*[*OF ε h*] **.**
with *h* **have** *current* (Γ ⊖ *ε*) (*plus-current h k*) **by**(*rule ε.current-plus-current-minus*)
hence *current* (Γ ⊖ *ε*) (*plus-current* (*plus-current h k*) *?g*) **using** *g* **unfolding** *minus-plus-current*[*OF f k*]
unfolding *F-simps minus-plus-current*[*OF ε h*] *ε.minus-plus-current*[*OF h k′*, *symmetric*]
by(*rule ε.current-plus-current-minus*)
then have *current* (Γ ⊖ *ε* ⊖ *?g*) (*plus-current* (*plus-current h k*) *?g* − *?g*) **using** *g′*
by(*rule ε.current-minus*)(*auto simp add*: *add-increasing*)
then have *h″*: *current* (Γ ⊖ *?ε′*) *?h′*
by(*rule arg-cong2*[**where** *f*=*current*, *THEN iffD1*, *rotated* −*1*])
(*simp-all add*: *minus-plus-current*[*OF ε g′*] *fun-eq-iff add-diff-eq-ennreal*[*symmetric*])
moreover have *wave* (Γ ⊖ *?ε′*) *?h′*
proof
have *separating* (Γ ⊖ *ε*) ($TER_{\Gamma \ominus \varepsilon}$ (*plus-current h k*))
using *k k-w* **unfolding** *F-simps minus-plus-current*[*OF ε h*]
by(*intro waveD-separating ε.wave-plus-current-minus*[*OF h h-w*])
moreover have $TER_{\Gamma \ominus \varepsilon}$ (*plus-current h k*) ⊆ $TER_{\Gamma \ominus ?\varepsilon'}$ (*plus-current h k*)
by(*auto 4 4 simp add*: *SAT.simps weight-minus-web*[*OF ε*] *weight-minus-web*[*OF ε′*] *split*: *if-split-asm elim*: *order-trans*[*rotated*] *intro*!: *ennreal-minus-mono d-IN-mono add-increasing2 δ-nonneg*)
ultimately show *sep*: *separating* (Γ ⊖ *?ε′*) ($TER_{\Gamma \ominus ?\varepsilon'}$ *?h′*)
by(*simp add*: *minus-plus-current*[*OF ε g′*] *separating-weakening*)
qed(*rule h″*)
moreover
have ¬ *hindered* (Γ ⊖ *F* (*?ε′*, *?h′*)) **using** *unhindered′*
proof(*rule contrapos-nn*)
assume *hindered* (Γ ⊖ *F* (*?ε′*, *?h′*))
thus *hindered ?Ω*
proof(*rule hindered-mono-web*[*rotated* −*1*])
show *weight ?Ω z* = *weight* (Γ ⊖ *F* (*?ε′*, *?h′*)) *z* **if** $z \notin A$ (Γ ⊖ *F* (*?ε′*, *?h′*)) **for** *z*
using *that* **unfolding** *F′*
apply(*cases z* = *y*)

apply(*simp-all add*: Ω*-def minus-plus-current*[*OF fk g*] Γ'*.weight-minus-web*[*OF g*] *IN-g*)
apply(*simp-all add*: *plus-current-def d-IN-add diff-add-eq-diff-diff-swap-ennreal currentD-finite-IN*[*OF f*])
done
have $y \neq a$ **using** *y-B a disjoint* **by** *auto*
then show *weight* ($\Gamma \ominus F$ (*?ε′*, *?h′*)) $z \leq$ *weight* *?Ω* z **if** $z \in A$ ($\Gamma \ominus F$ (*?ε′*, *?h′*)) **for** z
using *that y-B disjoint δ-nonneg* **unfolding** F'
apply(*cases* $z = a$)
apply(*simp-all add*: Ω*-def minus-plus-current*[*OF fk g*] Γ'*.weight-minus-web*[*OF g*] *OUT-g*)
apply(*auto simp add*: *plus-current-def d-OUT-add diff-add-eq-diff-diff-swap-ennreal currentD-finite-OUT*[*OF f*])
done
qed(*simp-all add*: Ω*-def*)
qed
ultimately have (*?ε′*, *?h′*) $\in$ *Field leq* **by**−(*rule F-I*)
with *Pair le sat that* **show** *?thesis* **by**(*auto*)
next
case *False*
with *currentD-weight-OUT*[*OF fk*, *of a*] **have** *d-OUT ?fk a* = *weight* Γ *a* **by** *simp*
have *sat εh* = *εh* **using** *False Pair* **by**(*simp add*: *sat-def k-def*)
thus *?thesis* **using** *that Pair* **by**(*auto*)
qed
qed

have *bourbaki-witt-fixpoint Sup leq sat* **using** *increasing chain-Field* **unfolding** *leq-def*
by(*intro bourbaki-witt-fixpoint-restrict-rel*)(*auto intro*: *Sup-upper Sup-least*)
then interpret *bourbaki-witt-fixpoint Sup leq sat* **.**

define *f* **where** *f* = *fixp-above* (*zero-current*, *zero-current*)
have *Field*: *f* $\in$ *Field leq* **using** *fixp-above-Field*[*OF zero*] **unfolding** *f-def* **.**
then have *f*: *current* Γ (*F f*) **and** *unhindered*: $\neg$ *hindered* ($\Gamma \ominus F f$)
by(*cases f*; *simp add*: *f unhindered′*; *fail*)+
interpret Γ: *countable-bipartite-web* $\Gamma \ominus F f$ **using** *f* **by**(*rule countable-bipartite-web-minus-web*)
note [*simp*] = *weight-minus-web*[*OF f*]
have *Field′*: (*fst f*, *snd f*) $\in$ *Field leq* **using** *Field* **by** *simp*

let *?P-k* = λk. *current* ($\Gamma \ominus F f$) *k* $\wedge$ *wave* ($\Gamma \ominus F f$) *k* $\wedge$ ($\forall k'$. *current* ($\Gamma \ominus F f$) k' $\wedge$ *wave* ($\Gamma \ominus F f$) k' $\wedge$ $k \leq k' \longrightarrow k = k'$)
define *k* **where** *k* = *Eps ?P-k*
have *Ex ?P-k* **by**(*intro ex-maximal-wave*)(*simp-all*)
hence *?P-k k* **unfolding** *k-def* **by**(*rule someI-ex*)
hence *k*: *current* ($\Gamma \ominus F f$) *k* **and** *k-w*: *wave* ($\Gamma \ominus F f$) *k*
and *maximal*: $\bigwedge k'$. $\llbracket$ *current* ($\Gamma \ominus F f$) k'; *wave* ($\Gamma \ominus F f$) k'; $k \leq k'$ $\rrbracket \Longrightarrow k = k'$ **by** *blast*+

note [*simp*] = Γ.*weight-minus-web*[*OF k*]

let *?fk* = *plus-current* (*F f*) *k*
have *IN-fk*: *d-IN ?fk x* = *d-IN* (*F f*) *x* + *d-IN k x* **for** *x*
by(*simp add*: *d-IN-def nn-integral-add*)
have *OUT-fk*: *d-OUT ?fk x* = *d-OUT* (*F f*) *x* + *d-OUT k x* **for** *x*
by(*simp add*: *d-OUT-def nn-integral-add*)
have *fk*: *current* Γ *?fk* **using** *f k* **by**(*rule current-plus-current-minus*)

have *d-OUT ?fk a* ≥ *weight* Γ *a*
proof(*rule ccontr*)
assume ¬ *?thesis*
hence *less*: *d-OUT ?fk a* < *weight* Γ *a* **by** *simp*

define Ω **where** Ω = Γ ⊖ *F f* ⊖ *k*
have *B*-Ω [*simp*]: *B* Ω = *B* Γ **by**(*simp add*: Ω-*def*)

have *loose*: *loose* Ω **unfolding** Ω-*def* **using** *unhindered k k-w maximal* **by**(*rule* Γ.*loose-minus-web*)
interpret Ω: *countable-bipartite-web* Ω **using** *k* **unfolding** Ω-*def*
by(*rule* Γ.*countable-bipartite-web-minus-web*)

have *a*-$\mathcal{E}$: *a* ∈ $TER_{\Gamma \ominus F\ f}$ *k* **using** *Field k k-w less* **by**(*rule a-TER*)
then have *weight* Ω *a* = *weight* Γ *a* − *d-OUT* (*F f*) *a*
using *a disjoint* **by**(*auto simp add*: *roofed-circ-def* Ω-*def SINK.simps*)
then have *weight-a*: *0* < *weight* Ω *a* **using** *less a*-$\mathcal{E}$
by(*simp add*: *OUT-fk SINK.simps diff-gr0-ennreal*)

let *?P-y* = λ*y*. *y* ∈ $\mathbf{OUT}_{\Omega}$ *a* ∧ *weight* Ω *y* > *0*
define *y* **where** *y* = *Eps ?P-y*
have *Ex ?P-y* **using** *Field k k-w less* **unfolding** Ω-*def* **by**(*rule Ex-y*)
hence *?P-y y* **unfolding** *y-def* **by**(*rule someI-ex*)
hence *y* ∈ $\mathbf{OUT}_{\Omega}$ *a* **and** *weight-y*: *weight* Ω *y* > *0* **by** *blast*+
then have *y-B*: *y* ∈ *B* Ω **by**(*auto simp add*: *outgoing-def* Ω-*def dest*: *bipartite-E*)

define *bound* **where** *bound* = *enn2real* (*min* (*weight* Ω *a*) (*weight* Ω *y*))
have *bound-pos*: *bound* > *0* **using** *weight-y weight-a* Ω.*weight-finite*
by(*cases weight* Ω *a weight* Ω *y rule*: *ennreal2-cases*)
(*simp-all add*: *bound-def min-def split*: *if-split-asm*)

let *?P*-δ = λδ. δ > *0* ∧ δ < *bound* ∧ ¬ *hindered* (*reduce-weight* Ω *y* δ)
define δ **where** δ = *Eps ?P*-δ
from Ω.*unhinder*[*OF loose - weight-y bound-pos*] *y-B disjoint* **have** *Ex ?P*-δ **by**(*auto simp add*: Ω-*def*)
hence *?P*-δ δ **unfolding** δ-*def* **by**(*rule someI-ex*)
hence δ-*pos*: *0* < δ **by** *blast*+

let *?f′* = (*plus-current* (*fst f*) (*zero-current*((*a*, *y*) := δ)), *plus-current* (*snd f*) *k*)

have *sat*: *?f′* = *sat f* **using** *less* **by**(*simp add*: *sat-def k-def Ω-def Let-def y-def bound-def δ-def split-def*)
also have ... = *f* **unfolding** *f-def* **using** *fixp-above-unfold*[*OF zero*] **by** *simp*
finally have *fst ?f′* (*a*, *y*) = *fst f* (*a*, *y*) **by** *simp*
hence *δ* = *0* **using** *currentD-finite*[*OF ε-curr*[*OF Field′*]] *δ-pos*
by(*cases fst f* (*a*, *y*)) *simp-all*
with *δ-pos* **show** *False* **by** *simp*
qed

with *currentD-weight-OUT*[*OF fk*, *of a*] **have** *d-OUT ?fk a* = *weight* Γ *a* **by** *simp*
moreover have *current* Γ *?fk* **using** *f k* **by**(*rule current-plus-current-minus*)
moreover have ¬ *hindered* (Γ ⊖ *?fk*) **unfolding** *minus-plus-current*[*OF f k*]
using *unhindered k k-w* **by**(*rule* Γ.*unhindered-minus-web*)
ultimately show *?thesis* **by** *blast*
qed

end

10.5 Linkability of unhindered bipartite webs

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

theorem *unhindered-linkable*:
assumes *unhindered*: ¬ *hindered* Γ
shows *linkable* Γ
proof(*cases A* Γ = {})
case *True*
thus *?thesis* **by**(*auto intro*!: *exI*[**where** *x=zero-current*] *linkage.intros simp add*: *web-flow-iff*)
next
case *nempty*: *False*

let *?P* = *λf a f′*. *current* (Γ ⊖ *f*) *f′* ∧ *d-OUT f′ a* = *weight* (Γ ⊖ *f*) *a* ∧ ¬ *hindered* (Γ ⊖ *f* ⊖ *f′*)

define *enum* **where** *enum* = *from-nat-into* (*A* Γ)
have *enum-A*: *enum n* ∈ *A* Γ **for** *n* **using** *from-nat-into*[*OF nempty*, *of n*] **by**(*simp add*: *enum-def*)
have *vertex-enum* [*simp*]: *vertex* Γ (*enum n*) **for** *n* **using** *enum-A*[*of n*] *A-vertex* **by** *blast*

define *f* **where** *f* = *rec-nat zero-current* (*λn f*. *let f′* = *SOME f′*. *?P f* (*enum n*) *f′ in plus-current f f′*)
have *f-0* [*simp*]: *f 0* = *zero-current* **by**(*simp add*: *f-def*)
have *f-Suc*: *f* (*Suc n*) = *plus-current* (*f n*) (*Eps* (*?P* (*f n*) (*enum n*))) **for** *n* **by**(*simp add*: *f-def*)

have *f*: *current* Γ (*f n*)

and *sat*: $\bigwedge m.\ m < n \Longrightarrow$ *d-OUT* (*f n*) (*enum m*) = *weight* Γ (*enum m*)
and *unhindered*: $\neg$ *hindered* ($\Gamma \ominus$ *f n*) **for** *n*
proof(*induction n*)
case *0*
{ case *1* **thus** *?case* **by** *simp* **}**
{ case *2* **thus** *?case* **by** *simp* **}**
{ case *3* **thus** *?case* **using** *unhindered* **by** *simp* **}**
next
case (*Suc n*)
interpret Γ: *countable-bipartite-web* $\Gamma \ominus$ *f n* **using** *Suc.IH*(*1*) **by**(*rule countable-bipartite-web-minus-web*)

define f' **where** f' = *Eps* (*?P* (*f n*) (*enum n*))
have *Ex* (*?P* (*f n*) (*enum n*)) **using** *Suc.IH*(*3*) **by**(*rule* Γ.*unhindered-saturate1*)(*simp add*: *enum-A*)
hence *?P* (*f n*) (*enum n*) f' **unfolding** f'*-def* **by**(*rule someI-ex*)
hence f': *current* ($\Gamma \ominus$ *f n*) f'
and *OUT*: *d-OUT* f' (*enum n*) = *weight* ($\Gamma \ominus$ *f n*) (*enum n*)
and *unhindered'*: $\neg$ *hindered* ($\Gamma \ominus$ *f n* $\ominus$ f') **by** *blast+*
have *f-Suc*: *f* (*Suc n*) = *plus-current* (*f n*) f' **by**(*simp add*: f'*-def f-Suc*)
{ case *1* **show** *?case* **unfolding** *f-Suc* **using** *Suc.IH*(*1*) f' **by**(*rule current-plus-current-minus*) **}**
note f'' = *this*
{ case (*2 m*)
have *d-OUT* (*f* (*Suc n*)) (*enum m*) $\leq$ *weight* Γ (*enum m*) **using** f'' **by**(*rule currentD-weight-OUT*)
moreover have *weight* Γ (*enum m*) $\leq$ *d-OUT* (*f* (*Suc n*)) (*enum m*)
proof(*cases m = n*)
case *True*
then show *?thesis* **unfolding** *f-Suc* **using** *OUT True*
by(*simp add*: *d-OUT-def nn-integral-add enum-A add-diff-self-ennreal less-imp-le*)
next
case *False*
hence $m < n$ **using** *2* **by** *simp*
thus *?thesis* **using** *Suc.IH*(*2*)[*OF* ‹$m < n$›] **unfolding** *f-Suc*
by(*simp add*: *d-OUT-def nn-integral-add add-increasing2*)
qed
ultimately show *?case* **by**(*rule antisym*) **}**
{ case *3* **show** *?case* **unfolding** *f-Suc minus-plus-current*[*OF Suc.IH*(*1*) f'] **by**(*rule unhindered'*) **}**
qed
interpret Γ: *countable-bipartite-web* $\Gamma \ominus$ *f n* **for** *n* **using** *f* **by**(*rule countable-bipartite-web-minus-web*)

have *Ex-P*: *Ex* (*?P* (*f n*) (*enum n*)) **for** *n* **using** *unhindered* **by**(*rule* Γ.*unhindered-saturate1*)(*simp add*: *enum-A*)
have *f-mono*: *f n* $\leq$ *f* (*Suc n*) **for** *n* **using** *someI-ex*[*OF Ex-P*, *of n*]
by(*auto simp add*: *le-fun-def f-Suc enum-A intro*: *add-increasing2 dest*:)

hence *incseq*: *incseq f* **by**(*rule incseq-SucI*)
hence *chain*: *Complete-Partial-Order.chain* (≤) (*range f*) **by**(*rule incseq-chain-range*)

define *g* **where** *g* = *Sup* (*range f*)
have *support-flow g* ⊆ **E**
by (*auto simp add*: *g-def support-flow.simps currentD-outside* [*OF f*] *image-comp elim*: *contrapos-pp*)
then have *countable-g*: *countable* (*support-flow g*) **by**(*rule countable-subset*) *simp*
with *chain* - - **have** *g*: *current* Γ *g* **unfolding** *g-def* **by**(*rule current-Sup*)(*auto simp add*: *f*)
moreover
have *d-OUT g x* = *weight* Γ *x* **if** *x* ∈ *A* Γ **for** *x*
proof(*rule antisym*)
show *d-OUT g x* ≤ *weight* Γ *x* **using** *g* **by**(*rule currentD-weight-OUT*)
have *countable* (*A* Γ) **using** *A-vertex* **by**(*rule countable-subset*) *simp*
from *that subset-range-from-nat-into*[*OF this*] **obtain** *n* **where** *x* = *enum n* **unfolding** *enum-def* **by** *blast*
with *sat*[*of n Suc n*] **have** *d-OUT* (*f* (*Suc n*)) *x* ≥ *weight* Γ *x* **by** *simp*
then show *weight* Γ *x* ≤ *d-OUT g x* **using** *countable-g* **unfolding** *g-def*
by(*subst d-OUT-Sup*[*OF chain*])(*auto intro*: *SUP-upper2*)
qed
ultimately show *?thesis* **by**(*auto simp add*: *web-flow-iff linkage.simps*)
qed

end

context *countable-web* **begin**

theorem *loose-linkable*: — Theorem 6.2
assumes *loose* Γ
shows *linkable* Γ
proof −
interpret *bi*: *countable-bipartite-web bipartite-web-of* Γ **by**(*rule countable-bipartite-web-of*)
have ¬ *hindered* (*bipartite-web-of* Γ) **using** *assms* **by**(*rule unhindered-bipartite-web-of*)
then have *linkable* (*bipartite-web-of* Γ)
by(*rule bi.unhindered-linkable*)
then show *?thesis* **by**(*rule linkable-bipartite-web-ofD*) *simp*
qed

lemma *ex-orthogonal-current*: — Lemma 4.15
∃ *f S*. *web-flow* Γ *f* ∧ *separating* Γ *S* ∧ *orthogonal-current* Γ *f S*
by(*rule ex-orthogonal-current'*)(*rule countable-web.loose-linkable*[*OF countable-web-quotient-web*])

end

10.6 Glueing the reductions together

context *countable-network* **begin**

context begin

qualified lemma *max-flow-min-cut'*:
  assumes *source-in*: $\bigwedge x.\ \neg\ edge\ \Delta\ x\ (source\ \Delta)$
  and *sink-out*: $\bigwedge y.\ \neg\ edge\ \Delta\ (sink\ \Delta)\ y$
  and *undead*: $\bigwedge x\ y.\ edge\ \Delta\ x\ y \Longrightarrow (\exists z.\ edge\ \Delta\ y\ z) \lor (\exists z.\ edge\ \Delta\ z\ x)$
  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 $\exists f\ S.\ flow\ \Delta\ f \land cut\ \Delta\ S \land orthogonal\ \Delta\ f\ S$
  by(*rule max-flow-min-cut'*)(*rule countable-web.ex-orthogonal-current*[*OF countable-web-web-of-network*], *fact+*)

qualified lemma *max-flow-min-cut''*:
  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 $\exists f\ S.\ flow\ \Delta\ f \land cut\ \Delta\ S \land orthogonal\ \Delta\ f\ S$
proof −
  interpret *antiparallel-edges* Δ **..**
  interpret Δ'': *countable-network* Δ'' **by**(*rule* Δ''*-countable-network*)
  have $\exists f\ S.\ flow\ \Delta''\ f \land cut\ \Delta''\ S \land orthogonal\ \Delta''\ f\ S$
    by(*rule* Δ''*.max-flow-min-cut'*)(*auto simp add*: *sink-out source-in no-loop capacity-pos elim*: *edg.cases*)
  then obtain *f S* **where** *f*: *flow* Δ'' *f* **and** *cut*: *cut* Δ'' *S* **and** *ortho*: *orthogonal* Δ'' *f S* **by** *blast*
  have *flow* Δ (*collect f*) **using** *f* **by**(*rule flow-collect*)
  moreover have *cut* Δ (*cut' S*) **using** *cut* **by**(*rule cut-cut'*)
  moreover have *orthogonal* Δ (*collect f*) (*cut' S*) **using** *ortho f* **by**(*rule orthogonal-cut'*)
  ultimately show *?thesis* **by** *blast*
qed

qualified lemma *max-flow-min-cut'''*:
  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 $\exists f\ S.\ flow\ \Delta\ f \land cut\ \Delta\ S \land orthogonal\ \Delta\ f\ S$
proof −
  interpret *antiparallel-edges* Δ **..**
  interpret Δ'': *countable-network* Δ'' **by**(*rule* Δ''*-countable-network*)
  have $\exists f\ S.\ flow\ \Delta''\ f \land cut\ \Delta''\ S \land orthogonal\ \Delta''\ f\ S$
    by(*rule* Δ''*.max-flow-min-cut''*)(*auto simp add*: *sink-out source-in capacity-pos elim*: *edg.cases*)
  then obtain *f S* **where** *f*: *flow* Δ'' *f* **and** *cut*: *cut* Δ'' *S* **and** *ortho*: *orthogonal* Δ'' *f S* **by** *blast*
  have *flow* Δ (*collect f*) **using** *f* **by**(*rule flow-collect*)

moreover have *cut* Δ (*cut′ S*) **using** *cut* **by**(*rule cut-cut′*)
moreover have *orthogonal* Δ (*collect f*) (*cut′ S*) **using** *ortho f* **by**(*rule orthogonal-cut′*)
ultimately show *?thesis* **by** *blast*
qed

theorem *max-flow-min-cut*:
$\exists f\, S.$ *flow* Δ *f* $\wedge$ *cut* Δ *S* $\wedge$ *orthogonal* Δ *f S*
proof −
interpret Δ''': *countable-network* Δ''' **by**(*rule* Δ'''*-countable-network*)
have $\exists f\, S.$ *flow* Δ''' *f* $\wedge$ *cut* Δ''' *S* $\wedge$ *orthogonal* Δ''' *f S* **by**(*rule* Δ'''*.max-flow-min-cut′′*) *auto*
then obtain *f S* **where** *f*: *flow* Δ''' *f* **and** *cut*: *cut* Δ''' *S* **and** *ortho*: *orthogonal* Δ''' *f S* **by** *blast*
from *flow-*Δ'''[*OF this*] **show** *?thesis* **by** *blast*
qed

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* Δ
and *capacity-outside*: $\forall e.\ e \notin \mathbf{E} \longrightarrow$ *capacity* Δ $e = 0$
and *capacity-finite* [*simp*]: $\forall e.$ *capacity* Δ $e \neq \top$
shows $\exists f\, S.$ *flow* Δ *f* $\wedge$ *cut* Δ *S* $\wedge$ *orthogonal* Δ *f S*
proof −
interpret *countable-network* Δ **using** *assms* **by**(*unfold-locales*) *auto*
show *?thesis* **by**(*rule max-flow-min-cut*)
qed

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-pmf* p) $A \leq$ *measure* (*measure-pmf* q) $\{y.\ \exists x{\in}A.\ R\ x\ y\}$
shows *rel-pmf* R p q
proof −
let $?A$ = *set-pmf* p **and** $?f = \lambda x.$ *ennreal* (*pmf* p x)
and $?B$ = *set-pmf* q **and** $?g = \lambda y.$ *ennreal* (*pmf* q y)
define R' **where** $R' = \{(x, y){\in}?A \times ?B.\ R\ x\ y\}$

have $(\sum^+ x{\in}?A.\ ?f\ x) = (\sum^+ y{\in}?B.\ ?g\ y)$ (**is** $?lhs = ?rhs$)
and $(\sum^+ y{\in}?B.\ ?g\ y) \neq \top$ (**is** *?bounded*)
proof −
have $?lhs = (\sum^+ x.\ ?f\ x)$ $?rhs = (\sum^+ y.\ ?g\ y)$
by(*auto simp add*: *nn-integral-count-space-indicator pmf-eq-0-set-pmf intro*!: *nn-integral-cong split*: *split-indicator*)
then show $?lhs = ?rhs$ *?bounded* **by**(*simp-all add*: *nn-integral-pmf-eq-1*)
qed
moreover
have $(\sum^+ x{\in}X.\ ?f\ x) \leq (\sum^+ y{\in}R'\ ``\ X.\ ?g\ y)$ (**is** $?lhs \leq ?rhs$) **if** $X \subseteq$ *set-pmf* p **for** X
proof −
have $?lhs$ = *measure* (*measure-pmf* p) X
by(*simp add*: *nn-integral-pmf measure-pmf.emeasure-eq-measure*)
also have $\ldots \leq$ *measure* (*measure-pmf* q) $\{y.\ \exists x{\in}X.\ R\ x\ y\}$ **by**(*simp add*: *le*)
also have $\ldots =$ *measure* (*measure-pmf* q) $(R'\ ``\ X)$ **using** *that*
by(*auto 4 3 simp add*: R'*-def AE-measure-pmf-iff intro*!: *measure-eq-AE*)
also have $\ldots = ?rhs$ **by**(*simp add*: *nn-integral-pmf measure-pmf.emeasure-eq-measure*)
finally show *?thesis* **.**
qed
moreover have *countable ?A countable ?B* **by** *simp-all*
moreover have $R' \subseteq ?A \times ?B$ **by**(*auto simp add*: R'*-def*)
ultimately obtain h
where *supp*: $\bigwedge x\ y.\ 0 < h\ x\ y \Longrightarrow (x, y) \in R'$
and *bound*: $\bigwedge x\ y.\ h\ x\ y \neq \top$
and *p*: $\bigwedge x.\ x \in ?A \Longrightarrow (\sum^+ y{\in}?B.\ h\ x\ y) = ?f\ x$
and *q*: $\bigwedge y.\ y \in ?B \Longrightarrow (\sum^+ x{\in}?A.\ h\ x\ y) = ?g\ y$
by(*rule bounded-matrix-for-marginals-ennreal*) *blast+*

let $?z = \lambda(x, y).$ *enn2real* $(h\ x\ y)$
define z **where** z = *embed-pmf* $?z$
have *nonneg*: $\bigwedge xy.\ 0 \leq ?z\ xy$ **by** *clarsimp*
have *outside*: $h\ x\ y = 0$ **if** $x \notin$ *set-pmf* $p \lor y \notin$ *set-pmf* $q \lor \neg\ R\ x\ y$ **for** x y
using *supp*[*of* x y] *that* **by**(*cases* $h\ x\ y > 0$)(*auto simp add*: R'*-def*)

have *prob*: $(\sum^+ xy.\ ?z\ xy) = 1$
proof −
have $(\sum^+ xy.\ ?z\ xy) = (\sum^+ x.\ \sum^+ y.\ (ennreal \circ ?z)\ (x,\ y))$
unfolding *nn-integral-fst-count-space* **by**(*simp add*: *split-def o-def*)
also have $\ldots = (\sum^+ x.\ (\sum^+ y.\ h\ x\ y))$ **using** *bound*
by(*simp add*: *nn-integral-count-space-reindex ennreal-enn2real-if*)
also have $\ldots = (\sum^+ x{\in}?A.\ (\sum^+ y{\in}?B.\ h\ x\ y))$
by(*auto intro*!: *nn-integral-cong nn-integral-zero′ simp add*: *nn-integral-count-space-indicator outside split*: *split-indicator*)
also have $\ldots = (\sum^+ x{\in}?A.\ ?f\ x)$ **by**(*auto simp add*: *p intro*!: *nn-integral-cong*)
also have $\ldots = (\sum^+ x.\ ?f\ x)$
by(*auto simp add*: *nn-integral-count-space-indicator pmf-eq-0-set-pmf intro*!: *nn-integral-cong split*: *split-indicator*)
finally show *?thesis* **by**(*simp add*: *nn-integral-pmf-eq-1*)
qed
note z = *nonneg prob*
have *z-sel* [*simp*]: *pmf* $z\ (x,\ y)$ = *enn2real* $(h\ x\ y)$ **for** $x\ y$
by(*simp add*: *z-def pmf-embed-pmf*[*OF z*])

show *?thesis*
proof
show $R\ x\ y$ **if** $(x,\ y) \in$ *set-pmf z* **for** $x\ y$ **using** *that*
using *that outside*[*of x y*] **unfolding** *set-pmf-iff*
by(*auto simp add*: *enn2real-eq-0-iff*)

show *map-pmf fst z* = p
proof(*rule pmf-eqI*)
fix x
have *pmf* (*map-pmf fst z*) $x = (\sum^+ e{\in}range\ (Pair\ x).\ pmf\ z\ e)$
by(*auto simp add*: *ennreal-pmf-map nn-integral-measure-pmf nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots = (\sum^+ y.\ h\ x\ y)$
using *bound* **by**(*simp add*: *nn-integral-count-space-reindex ennreal-enn2real-if*)
also have $\ldots = (\sum^+ y{\in}?B.\ h\ x\ y)$ **using** *outside*
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots = ?f\ x$ **using** p[*of x*] **apply**(*cases* $x \in$ *set-pmf p*)
by(*auto simp add*: *set-pmf-iff AE-count-space outside intro*!: *nn-integral-zero′*)
finally show *pmf* (*map-pmf fst z*) x = *pmf p x* **by** *simp*
qed

show *map-pmf snd z* = q
proof(*rule pmf-eqI*)
fix y
have *pmf* (*map-pmf snd z*) $y = (\sum^+ e{\in}range\ (\lambda x.\ (x,\ y)).\ pmf\ z\ e)$
by(*auto simp add*: *ennreal-pmf-map nn-integral-measure-pmf nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots = (\sum^+ x.\ h\ x\ y)$
using *bound* **by**(*simp add*: *nn-integral-count-space-reindex ennreal-enn2real-if*)

also have ... = ($\sum^+ x \in ?A.\ h\ x\ y$) **using** *outside*
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *?g y* **using** *q*[*of y*] **apply**(*cases y* ∈ *set-pmf q*)
by(*auto simp add*: *set-pmf-iff AE-count-space outside intro*!: *nn-integral-zero′*)
finally show *pmf* (*map-pmf snd z*) *y* = *pmf q y* **by** *simp*
qed
qed
qed

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-pmf p* ⟹ *measure-pmf.prob p A* ≤ *measure-pmf.prob q* {*y* ∈ *set-pmf q*. ∃ *x*∈*A*. *R x y*}
shows *rel-pmf R p q*
proof(*rule rel-pmf-measureI*)
fix *A*
let *?A* = *A* ∩ *set-pmf p*
have *measure-pmf.prob p A* = *measure-pmf.prob p ?A* **by**(*simp add*: *measure-Int-set-pmf*)
also have ... ≤ *measure-pmf.prob q* {*y* ∈ *set-pmf q*. ∃ *x*∈*?A*. *R x y*} **by**(*rule le*) *simp*
also have ... ≤ *measure-pmf.prob q* {*y*. ∃ *x*∈*A*. *R x y*}
by(*rule measure-pmf.finite-measure-mono*) *auto*
finally show *measure-pmf.prob p A* ≤ ... **.**
qed

lemma *rel-pmf-code* [*code*]:
rel-pmf R p q ⟷
(*let B* = *set-pmf q in*
∀ *A*∈*Pow* (*set-pmf p*). *measure-pmf.prob p A* ≤ *measure-pmf.prob q* (*snd* ‘ *Set.filter* (*case-prod R*) (*A* × *B*)))
unfolding *Let-def*
proof(*intro iffI strip*)
have *eq*: *snd* ‘ *Set.filter* (*case-prod R*) (*A* × *set-pmf q*) = {*y*. ∃ *x*∈*A*. *R x y*} ∩ *set-pmf q* **for** *A*
by (*rule set-eqI*) (*auto simp add*: *image-iff*)
show *measure-pmf.prob p A* ≤ *measure-pmf.prob q* (*snd* ‘ *Set.filter* (*case-prod R*) (*A* × *set-pmf q*))
if *rel-pmf R p q* **and** *A* ∈ *Pow* (*set-pmf p*) **for** *A*
using *that* **by** (*simp only*: *eq*)(*auto dest*: *rel-pmf-measureD simp add*: *measure-Int-set-pmf*)
show *rel-pmf R p q* **if** ∀ *A*∈*Pow* (*set-pmf p*). *measure-pmf.prob p A* ≤ *measure-pmf.prob q* (*snd* ‘ *Set.filter* (*case-prod R*) (*A* × *set-pmf q*))
using *that* **by** (*intro rel-pmf-measureI′*) (*simp only*: *eq*, *auto intro*: *ord-le-eq-trans arg-cong2*[**where** *f*=*measure*])
qed

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*
by(*simp add*: *inj-on-def*)

private lemma *inj-Right* [*simp*]: *inj-on Right X*
by(*simp add*: *inj-on-def*)

context fixes *p* :: 'a *pmf* **and** *q* :: 'b *pmf* **and** *R* :: 'a ⇒ 'b ⇒ *bool* **begin**

private inductive *edge'* :: ('a, 'b) *vertex* ⇒ ('a, 'b) *vertex* ⇒ *bool* **where**
edge' Source (*Left x*) **if** *x* ∈ *set-pmf p*
| *edge'* (*Left x*) (*Right y*) **if** *R x y x* ∈ *set-pmf p y* ∈ *set-pmf q*
| *edge'* (*Right y*) *Sink* **if** *y* ∈ *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* ∈ *set-pmf p* ∧ *y* ∈ *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*
  by *pat-completeness auto*
termination by *lexicographic-order*

private 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*
  by(*simp-all add*: Δ-*def*)

private lemma *IN-Left* [*simp*]: $\mathbf{IN}_{\Delta}$ (*Left x*) = (*if x* $\in$ *set-pmf p then* {*Source*} *else* {})
  by(*auto simp add*: *incoming-def*)
private lemma *OUT-Right* [*simp*]: $\mathbf{OUT}_{\Delta}$ (*Right y*) = (*if y* $\in$ *set-pmf q then* {*Sink*} *else* {})
  by(*auto simp add*: *outgoing-def*)

interpretation *network*: *countable-network* Δ
proof
  show *source* $\Delta \neq$ *sink* Δ **by** *simp*
  show *capacity* Δ *e* = *0* **if** *e* $\notin \mathbf{E}_{\Delta}$ **for** *e* **using** *that*
    by(*cases e*; *cases fst e*; *cases snd e*)(*auto simp add*: *pmf-eq-0-set-pmf*)
  show *capacity* Δ *e* $\neq$ *top* **for** *e* **by**(*cases e rule*: *cap.cases*)(*auto*)
  have $\mathbf{E}_{\Delta} \subseteq$ ((*Pair Source* $\circ$ *Left*) ‘ *set-pmf p*) $\cup$ (*map-prod Left Right* ‘ (*set-pmf p* $\times$ *set-pmf q*)) $\cup$ ((λ*y*. (*Right y*, *Sink*)) ‘ *set-pmf q*)
    by(*auto elim*: *edge'.cases*)
  thus *countable* $\mathbf{E}_{\Delta}$ **by**(*rule countable-subset*) *auto*
qed

private lemma *OUT-cap-Source*: *d-OUT cap Source* = *1*
proof −
  have *d-OUT cap Source* = ($\sum^{+}$ *y*$\in$*range Left*. *cap* (*Source*, *y*))
    by(*auto 4 4 simp add*: *d-OUT-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.capacity-outside*[*simplified*] *split*: *split-indicator*)
  also have ... = ($\sum^{+}$ *y*. *pmf p y*) **by**(*simp add*: *nn-integral-count-space-reindex*)
  also have ... = *1* **by**(*simp add*: *nn-integral-pmf*)
  finally show *?thesis* **.**
qed

private lemma *IN-cap-Left*: *d-IN cap* (*Left x*) = *pmf p x*
by(*subst d-IN-alt-def*[*of* - Δ])(*simp-all add*: *pmf-eq-0-set-pmf nn-integral-count-space-indicator max-def*)
private lemma *OUT-cap-Right*: *d-OUT cap* (*Right y*) = *pmf q y*
by(*subst d-OUT-alt-def*[*of* - Δ])(*simp-all add*: *pmf-eq-0-set-pmf nn-integral-count-space-indicator max-def*)

private lemma *rel-pmf-measureI-aux*:
assumes *ex-flow*: $\exists f$ *S*. *flow* Δ *f* $\wedge$ *cut* Δ *S* $\wedge$ *orthogonal* Δ *f S*
and *le*: $\bigwedge$*A*. *measure* (*measure-pmf p*) *A* $\leq$ *measure* (*measure-pmf q*) {*y*. $\exists$ *x*$\in$*A*. *R x y*}
shows *rel-pmf R p q*
proof −
from *ex-flow* **obtain** *f S*
where *f*: *flow* Δ *f* **and** *cut*: *cut* Δ *S* **and** *ortho*: *orthogonal* Δ *f S* **by** *blast*
from *cut* **obtain** *Source*: *Source* $\in$ *S* **and** *Sink*: *Sink* $\notin$ *S* **by** *cases simp*

have *f-finite* [*simp*]: *f e* < *top* **for** *e*
using *network.flowD-finite*[*OF f*, *of e*] **by** (*simp-all add*: *less-top*)

have *IN-f-Left*: *d-IN f* (*Left x*) = *f* (*Source*, *Left x*) **for** *x*
by(*subst d-IN-alt-def*[*of* - Δ])(*simp-all add*: *nn-integral-count-space-indicator max-def network.flowD-outside*[*OF f*])
have *OUT-f-Right*: *d-OUT f* (*Right y*) = *f* (*Right y*, *Sink*) **for** *y*
by(*subst d-OUT-alt-def*[*of* - Δ])(*simp-all add*: *nn-integral-count-space-indicator max-def network.flowD-outside*[*OF f*])

have *value-flow* Δ *f* $\leq$ *1* **using** *flowD-capacity-OUT*[*OF f*, *of Source*] **by**(*simp add*: *OUT-cap-Source*)
moreover have *1* $\leq$ *value-flow* Δ *f*
proof −
let *?L* = *Left* −‘ *S* $\cap$ *set-pmf p*
let *?R′* = {*y*|*y x*. *x* $\in$ *set-pmf p* $\wedge$ *Left x* $\in$ *S* $\wedge$ *R x y* $\wedge$ *y* $\in$ *set-pmf q* $\wedge$ *Right y* $\in$ *S*}
let *?R″* = {*y*|*y x*. *x* $\in$ *set-pmf p* $\wedge$ *Left x* $\in$ *S* $\wedge$ *R x y* $\wedge$ *y* $\in$ *set-pmf q* $\wedge$ $\neg$ *Right y* $\in$ *S*}
have *value-flow* Δ *f* = ($\sum^+$ *x*$\in$*range Left*. *f* (*Source*, *x*)) **unfolding** *d-OUT-def*
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *split*: *split-indicator*)
also have ... = ($\sum^+$ *x*. *f* (*Source*, *Left x*) ∗ *indicator ?L x*) + ($\sum^+$ *x*. *f* (*Source*, *Left x*) ∗ *indicator* (− *?L*) *x*)
by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-cong split*: *split-indicator*)
also have ($\sum^+$ *x*. *f* (*Source*, *Left x*) ∗ *indicator* (− *?L*) *x*) = ($\sum^+$ *x*$\in$− *?L*. *cap* (*Source*, *Left x*))
using *orthogonalD-out*[*OF ortho* - *Source*]
apply(*auto simp add*: *set-pmf-iff network.flowD-outside*[*OF f*] *nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
subgoal for *x* **by**(*cases x* $\in$ *set-pmf p*)(*auto simp add*: *set-pmf-iff net-*

work.flowD-outside[*OF f*])
done
also have ... $= (\sum^+ x\in-\ ?L.\ pmf\ p\ x)$ **by** *simp*
also have ... $=$ *emeasure p* $(-\ ?L)$ **by**(*simp add*: *nn-integral-pmf*)
also have $(\sum^+ x.\ f\ (Source,\ Left\ x) * indicator\ ?L\ x) = (\sum^+ x\in ?L.\ d\text{-}IN\ f\ (Left\ x))$
by(*subst d-IN-alt-def*[*of - Δ*])(*auto simp add*: *network.flowD-outside*[*OF f*] *nn-integral-count-space-indicator intro*!: *nn-integral-cong*)
also have ... $= (\sum^+ x\in ?L.\ d\text{-}OUT\ f\ (Left\ x))$
by(*rule nn-integral-cong flowD-KIR*[*OF f*, *symmetric*])+ *simp-all*
also have ... $= (\sum^+ x.\ \sum^+ y.\ f\ (Left\ x,\ y) * indicator\ (range\ Right)\ y * indicator\ ?L\ x)$
by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *split*: *split-indicator*)
also have ... $= (\sum^+ y\in range\ Right.\ \sum^+ x.\ f\ (Left\ x,\ y) * indicator\ ?L\ x)$
by(*subst nn-integral-fst-count-space*[**where** *f=case-prod -*, *simplified*])
(*simp add*: *nn-integral-snd-count-space*[**where** *f=case-prod -*, *simplified*] *nn-integral-count-space-indicator nn-integral-cmult*[*symmetric*] *mult-ac*)
also have ... $= (\sum^+ y.\ \sum^+ x.\ f\ (Left\ x,\ Right\ y) * indicator\ ?L\ x)$
by(*simp add*: *nn-integral-count-space-reindex*)
also have ... $= (\sum^+ y.\ \sum^+ x.\ f\ (Left\ x,\ Right\ y) * indicator\ ?L\ x * indicator\ ?R'\ y) +$
$(\sum^+ y.\ \sum^+ x.\ f\ (Left\ x,\ Right\ y) * indicator\ ?L\ x * indicator\ ?R''\ y)$
by(*subst nn-integral-add*[*symmetric*]; *simp*)
(*subst nn-integral-add*[*symmetric*]; *auto intro*!: *nn-integral-cong split*: *split-indicator intro*!: *network.flowD-outside*[*OF f*])
also have $(\sum^+ y.\ \sum^+ x.\ f\ (Left\ x,\ Right\ y) * indicator\ ?L\ x * indicator\ ?R'\ y) =$
$(\sum^+ y.\ \sum^+ x.\ f\ (Left\ x,\ Right\ y) * indicator\ ?R'\ y)$
apply(*clarsimp simp add*: *network.flowD-outside*[*OF f*] *intro*!: *nn-integral-cong split*: *split-indicator*)
subgoal for *y x* **by**(*cases edge Δ* (*Left x*) (*Right y*))(*auto intro*: *orthogonalD-in*[*OF ortho*] *network.flowD-outside*[*OF f*])
done
also have ... $= (\sum^+ y.\ \sum^+ x\in range\ Left.\ f\ (x,\ Right\ y) * indicator\ ?R'\ y)$
by(*simp add*: *nn-integral-count-space-reindex*)
also have ... $= (\sum^+ y\in ?R'.\ d\text{-}IN\ f\ (Right\ y))$
by(*subst d-IN-alt-def*[*of - Δ*])(*auto simp add*: *network.flowD-outside*[*OF f*] *nn-integral-count-space-indicator incoming-def intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... $= (\sum^+ y\in ?R'.\ d\text{-}OUT\ f\ (Right\ y))$ **using** *flowD-KIR*[*OF f*] **by** *simp*
also have ... $= (\sum^+ y\in ?R'.\ d\text{-}OUT\ cap\ (Right\ y))$
by(*auto 4 3 intro*!: *nn-integral-cong simp add*: *d-OUT-def network.flowD-outside*[*OF f*] *Sink dest*: *intro*: *orthogonalD-out*[*OF ortho*, *of Right - Sink*, *simplified*])
also have ... $= (\sum^+ y\in ?R'.\ pmf\ q\ y)$ **by**(*simp add*: *OUT-cap-Right*)
also have ... $=$ *emeasure q ?R′* **by**(*simp add*: *nn-integral-pmf*)
also have $(\sum^+ y.\ \sum^+ x.\ f\ (Left\ x,\ Right\ y) * indicator\ ?L\ x * indicator\ ?R''\ y) \geq$ *emeasure q ?R″* (**is** *?lhs* ≥ -)
proof –

have *?lhs* = ($\sum^+$ *y*. $\sum^+$ *x*. (*if R x y then pmf q y else 0*) $*$ *indicator ?L x* $*$ *indicator ?R'' y*)
by(*rule nn-integral-cong*)+(*auto split*: *split-indicator simp add*: *network.flowD-outside*[*OF f*] *simp add*: *orthogonalD-out*[*OF ortho*, *of Left - Right -*, *simplified*])
also have ... $\geq$ ($\sum^+$ *y*. *pmf q y* $*$ *indicator ?R'' y*)
by(*rule nn-integral-mono*)(*auto split*: *split-indicator intro*: *order-trans*[*OF - nn-integral-ge-point*])
also have ($\sum^+$ *y*. *pmf q y* $*$ *indicator ?R'' y*) = ($\sum^+$ *y*∈*?R''*. *pmf q y*)
by(*auto simp add*: *nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
finally show *?thesis* **by**(*simp add*: *nn-integral-pmf*)
qed
ultimately have *value-flow* Δ *f* $\geq$ *emeasure q ?R'* + *emeasure q ?R''* + *emeasure p* (− *?L*)
by(*simp add*: *add-right-mono*)
also have *emeasure q ?R'* + *emeasure q ?R''* = *emeasure q* {*y*|*y x*. *x* ∈ *set-pmf p* ∧ *Left x* ∈ *S* ∧ *R x y* ∧ *y* ∈ *set-pmf q*}
by(*subst plus-emeasure*)(*auto intro*!: *arg-cong2*[**where** *f*=*emeasure*])
also have ... $\geq$ *emeasure p ?L* **using** *le*[*of ?L*]
by(*auto elim*!: *order-trans simp add*: *measure-pmf.emeasure-eq-measure AE-measure-pmf-iff intro*!: *measure-pmf.finite-measure-mono-AE*)
ultimately have *value-flow* Δ *f* $\geq$ *emeasure* (*measure-pmf p*) *?L* + *emeasure* (*measure-pmf p*) (− *?L*)
by (*smt* (*verit*, *best*) *add-right-mono inf.absorb-iff2 le-inf-iff*)
also have *emeasure* (*measure-pmf p*) *?L* + *emeasure* (*measure-pmf p*) (− *?L*) = *emeasure* (*measure-pmf p*) (*?L* ∪ − *?L*)
by(*subst plus-emeasure*) *auto*
also have *?L* ∪ −*?L* = *UNIV* **by** *blast*
finally show *?thesis* **by** *simp*
qed
ultimately have *val*: *value-flow* Δ *f* = *1* **by** *simp*

have *SAT-p*: *f* (*Source*, *Left x*) = *pmf p x* **for** *x*
proof(*rule antisym*)
show *f* (*Source*, *Left x*) $\leq$ *pmf p x* **using** *flowD-capacity*[*OF f*, *of* (*Source*, *Left x*)] **by** *simp*
show *pmf p x* $\leq$ *f* (*Source*, *Left x*)
proof(*rule ccontr*)
assume $*$: ¬ *?thesis*
have *finite*: ($\sum^+$ *y*. *f* (*Source*, *Left y*) $*$ *indicator* (− {*x*}) *y*) $\neq \infty$
proof −
have ($\sum^+$ *y*. *f* (*Source*, *Left y*) $*$ *indicator* (− {*x*}) *y*) $\leq$ ($\sum^+$ *y*∈*range Left*. *f* (*Source*, *y*))
by(*auto simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... = *value-flow* Δ *f*
by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *split*: *split-indicator*)
finally show *?thesis* **using** *val* **by** (*auto simp*: *top-unique*)

qed
have *value-flow* Δ f = ($\sum^+$ $y \in$*range Left*. f (*Source*, y))
by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *split*: *split-indicator*)
also have ... = ($\sum^+$ y. f (*Source*, *Left* y) $*$ *indicator* ($-$ $\{x\}$) y) + ($\sum^+$ y. f (*Source*, *Left* y) $*$ *indicator* $\{x\}$ y)
by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... < ($\sum^+$ y. f (*Source*, *Left* y) $*$ *indicator* ($-$ $\{x\}$) y) + ($\sum^+$ y. *pmf p y* $*$ *indicator* $\{x\}$ y) **using** $*$ *finite*
by(*auto simp add*:)
also have ... $\leq$ ($\sum^+$ y. *pmf p y* $*$ *indicator* ($-$ $\{x\}$) y) + ($\sum^+$ y. *pmf p y* $*$ *indicator* $\{x\}$ y)
using *flowD-capacity*[*OF f*, *of* (*Source*, *Left* -)]
by(*auto intro*!: *nn-integral-mono split*: *split-indicator*)
also have ... = ($\sum^+$ y. *pmf p y*)
by(*subst nn-integral-add*[*symmetric*])(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = *1* **unfolding** *nn-integral-pmf* **by** *simp*
finally show *False* **using** *val* **by** *simp*
qed
qed

have *IN-Sink*: *d-IN f Sink* = *1*
proof $-$
have *d-IN f Sink* = ($\sum^+$ $x \in$*range Right*. f (x, *Sink*)) **unfolding** *d-IN-def*
by(*auto intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *simp add*: *nn-integral-count-space-indicator split*: *split-indicator*)
also have ... = ($\sum^+$ y. *d-OUT f* (*Right y*)) **by**(*simp add*: *nn-integral-count-space-reindex OUT-f-Right*)
also have ... = ($\sum^+$ y. *d-IN f* (*Right y*)) **by**(*simp add*: *flowD-KIR*[*OF f*])
also have ... = ($\sum^+$ y. ($\sum^+$ $x \in$*range Left*. f (x, *Right y*)))
by(*auto simp add*: *d-IN-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *split*: *split-indicator*)
also have ... = ($\sum^+$ y. $\sum^+$ x. f (*Left x*, *Right y*)) **by**(*simp add*: *nn-integral-count-space-reindex*)
also have ... = ($\sum^+$ x. $\sum^+$ y. f (*Left x*, *Right y*))
by(*subst nn-integral-fst-count-space*[**where** *f*=*case-prod* -, *simplified*])(*simp add*: *nn-integral-snd-count-space*[**where** *f*=*case-prod* -, *simplified*])
also have ... = ($\sum^+$ x. ($\sum^+$ $y \in$*range Right*. f (*Left x*, y)))
by(*simp add*: *nn-integral-count-space-reindex*)
also have ... = ($\sum^+$ x. *d-OUT f* (*Left x*)) **unfolding** *d-OUT-def*
by(*auto intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *simp add*: *nn-integral-count-space-indicator split*: *split-indicator*)
also have ... = ($\sum^+$ x. *d-IN f* (*Left x*)) **by**(*simp add*: *flowD-KIR*[*OF f*])
also have ... = ($\sum^+$ x. *pmf p x*) **by**(*simp add*: *IN-f-Left SAT-p*)
also have ... = *1* **unfolding** *nn-integral-pmf* **by** *simp*
finally show *?thesis* **.**
qed

have *SAT-q*: f (*Right* y, *Sink*) $=$ *pmf* q y **for** y
proof(*rule antisym*)
show f (*Right* y, *Sink*) $\leq$ *pmf* q y **using** *flowD-capacity*[*OF* f, *of* (*Right* y, *Sink*)] **by** *simp*
show *pmf* q y $\leq$ f (*Right* y, *Sink*)
proof(*rule ccontr*)
assume $*$: $\neg$ *?thesis*
have *finite*: $(\sum^+ x.\ f$ (*Right* x, *Sink*) $*$ *indicator* $(-\ \{y\})\ x) \neq \infty$
proof $-$
have $(\sum^+ x.\ f$ (*Right* x, *Sink*) $*$ *indicator* $(-\ \{y\})\ x) \leq (\sum^+ x{\in}$*range Right*. f $(x$, *Sink*))
by(*auto simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-mono split*: *split-indicator*)
also have $\ldots$ $=$ *d-IN* f *Sink*
by(*auto simp add*: *d-IN-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF* f] *split*: *split-indicator*)
finally show *?thesis* **using** *IN-Sink* **by** (*auto simp*: *top-unique*)
qed
have *d-IN* f *Sink* $=$ $(\sum^+ x{\in}$*range Right*. f $(x$, *Sink*))
by(*auto simp add*: *d-IN-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF* f] *split*: *split-indicator*)
also have $\ldots$ $=$ $(\sum^+ x.\ f$ (*Right* x, *Sink*) $*$ *indicator* $(-\ \{y\})\ x) + (\sum^+ x.\ f$ (*Right* x, *Sink*) $*$ *indicator* $\{y\}\ x)$
by(*subst nn-integral-add*[*symmetric*])(*auto simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots$ $<$ $(\sum^+ x.\ f$ (*Right* x, *Sink*) $*$ *indicator* $(-\ \{y\})\ x) + (\sum^+ x.$ *pmf* q x $*$ *indicator* $\{y\}\ x)$ **using** $*$ *finite*
by *auto*
also have $\ldots$ $\leq$ $(\sum^+ x.$ *pmf* q x $*$ *indicator* $(-\ \{y\})\ x) + (\sum^+ x.$ *pmf* q x $*$ *indicator* $\{y\}\ x)$
using *flowD-capacity*[*OF* f, *of* (*Right* -, *Sink*)]
by(*auto intro*!: *nn-integral-mono split*: *split-indicator*)
also have $\ldots$ $=$ $(\sum^+ x.$ *pmf* q $x)$
by(*subst nn-integral-add*[*symmetric*])(*auto intro*!: *nn-integral-cong split*: *split-indicator*)
also have $\ldots$ $=$ *1* **unfolding** *nn-integral-pmf* **by** *simp*
finally show *False* **using** *IN-Sink* **by** *simp*
qed
qed

let *?z* $= \lambda(x, y).$ *enn2real* (f (*Left* x, *Right* y))
have *nonneg*: $\bigwedge xy.\ 0 \leq$ *?z xy* **by** *clarsimp*
have *prob*: $(\sum^+ xy.$ *?z xy*$) = 1$
proof $-$
have $(\sum^+ xy.$ *?z xy*$) = (\sum^+ x.\ \sum^+ y.$ (*ennreal* $\circ$ *?z*) $(x, y))$
unfolding *nn-integral-fst-count-space* **by**(*simp add*: *split-def o-def*)
also have $\ldots$ $=$ $(\sum^+ x.\ (\sum^+ y{\in}$*range Right*. f (*Left* x, y)))
by(*auto simp add*: *nn-integral-count-space-reindex intro*!: *nn-integral-cong*)
also have $\ldots$ $=$ $(\sum^+ x.$ *d-OUT* f (*Left* x))

by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator split*: *split-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*])
also have ... $=$ ($\sum^+$ *x*. *d-IN f* (*Left x*)) **using** *flowD-KIR*[*OF f*] **by** *simp*
also have ... $=$ ($\sum^+$ *x*∈*range Left*. *f* (*Source*, *x*)) **by**(*simp add*: *nn-integral-count-space-reindex IN-f-Left*)
also have ... $=$ *value-flow* Δ *f*
by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *split*: *split-indicator*)
finally show *?thesis* **using** *val* **by**(*simp*)
qed
note *z* = *nonneg prob*
define *z* **where** *z* = *embed-pmf ?z*
have *z-sel* [*simp*]: *pmf z* (*x*, *y*) = *enn2real* (*f* (*Left x*, *Right y*)) **for** *x y*
by(*simp add*: *z-def pmf-embed-pmf*[*OF z*])

show *?thesis*
proof
show *R x y* **if** (*x*, *y*) ∈ *set-pmf z* **for** *x y*
using *that network.flowD-outside*[*OF f*, *of* (*Left x*, *Right y*)] **unfolding** *set-pmf-iff*
by(*auto simp add*: *enn2real-eq-0-iff*)

show *map-pmf fst z* = *p*
proof(*rule pmf-eqI*)
fix *x*
have *pmf* (*map-pmf fst z*) *x* = ($\sum^+$ *e*∈*range* (*Pair x*). *pmf z e*)
by(*auto simp add*: *ennreal-pmf-map nn-integral-measure-pmf nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = ($\sum^+$ *y*∈*range Right*. *f* (*Left x*, *y*)) **by**(*simp add*: *nn-integral-count-space-reindex*)
also have ... = *d-OUT f* (*Left x*)
by(*auto simp add*: *d-OUT-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *split*: *split-indicator*)
also have ... = *d-IN f* (*Left x*) **by**(*rule flowD-KIR*[*OF f*]) *simp-all*
also have ... = *f* (*Source*, *Left x*) **by**(*simp add*: *IN-f-Left*)
also have ... = *pmf p x* **by**(*simp add*: *SAT-p*)
finally show *pmf* (*map-pmf fst z*) *x* = *pmf p x* **by** *simp*
qed

show *map-pmf snd z* = *q*
proof(*rule pmf-eqI*)
fix *y*
have *pmf* (*map-pmf snd z*) *y* = ($\sum^+$ *e*∈*range* (λ*x*. (*x*, *y*)). *pmf z e*)
by(*auto simp add*: *ennreal-pmf-map nn-integral-measure-pmf nn-integral-count-space-indicator intro*!: *nn-integral-cong split*: *split-indicator*)
also have ... = ($\sum^+$ *x*∈*range Left*. *f* (*x*, *Right y*)) **by**(*simp add*: *nn-integral-count-space-reindex*)
also have ... = *d-IN f* (*Right y*)
by(*auto simp add*: *d-IN-def nn-integral-count-space-indicator intro*!: *nn-integral-cong network.flowD-outside*[*OF f*] *split*: *split-indicator*)
also have ... = *d-OUT f* (*Right y*) **by**(*simp add*: *flowD-KIR*[*OF f*])

also have ... $=$ *f* (*Right y*, *Sink*) **by**(*simp add*: *OUT-f-Right*)
also have ... $=$ *pmf q y* **by**(*simp add*: *SAT-q*)
finally show *pmf* (*map-pmf snd z*) *y* $=$ *pmf q y* **by** *simp*
qed
qed
qed

proposition *rel-pmf-measureI-unbounded*: — Proof uses the unbounded max-flow min-cut theorem
assumes *le*: $\bigwedge A.$ *measure* (*measure-pmf p*) *A* $\leq$ *measure* (*measure-pmf q*) $\{y.\ \exists x \in A.\ R\ x\ y\}$
shows *rel-pmf R p q*
using *assms* **by**(*rule rel-pmf-measureI-aux*[*OF network.max-flow-min-cut*])

interpretation *network*: *bounded-countable-network* Δ
proof
have *OUT-Left*: *d-OUT cap* (*Left x*) $\leq$ *1* **for** *x*
proof $-$
have *d-OUT cap* (*Left x*) $\leq$ ($\sum^+$ *y*$\in$*range Right*. *cap* (*Left x*, *y*))
by(*subst d-OUT-alt-def*[*of* - Δ])(*auto intro*: *network.capacity-outside*[*simplified*] *intro*!: *nn-integral-mono simp add*: *nn-integral-count-space-indicator outgoing-def split*: *split-indicator*)
also have ... $=$ ($\sum^+$ *y*. *cap* (*Left x*, *Right y*)) **by**(*simp add*: *nn-integral-count-space-reindex*)
also have ... $\leq$ ($\sum^+$ *y*. *pmf q y*) **by**(*rule nn-integral-mono*)(*simp*)
also have ... $=$ *1* **by**(*simp add*: *nn-integral-pmf*)
finally show *?thesis* **.**
qed
show *d-OUT* (*capacity* Δ) *x* $< \top$ **if** $x \in \mathbf{V}_\Delta$ $x \neq$ *source* Δ $x \neq$ *sink* Δ **for** *x*
using *that* **by**(*cases x*)(*auto simp add*: *OUT-cap-Right intro*: *le-less-trans*[*OF OUT-Left*])
qed

proposition *rel-pmf-measureI-bounded*: — Proof uses the bounded max-flow min-cut theorem
assumes *le*: $\bigwedge A.$ *measure* (*measure-pmf p*) *A* $\leq$ *measure* (*measure-pmf q*) $\{y.\ \exists x \in A.\ R\ x\ y\}$
shows *rel-pmf R p q*
using *assms* **by**(*rule rel-pmf-measureI-aux*[*OF network.max-flow-min-cut-bounded*])

end

end

interpretation *rel-spmf-characterisation* **by** *unfold-locales*(*rule rel-pmf-measureI-bounded*)

corollary *rel-pmf-distr-mono*: *rel-pmf R OO rel-pmf S* $\leq$ *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(*intro le-funI le-boolI rel-pmf-measureI-bounded*, *elim relcomppE*)

fix *p q r A*
assume *pq*: *rel-pmf R p q* **and** *qr*: *rel-pmf S q r*
have *measure* (*measure-pmf p*) *A* ≤ *measure* (*measure-pmf q*) {*y*. ∃ *x*∈*A*. *R x y*}
(**is** - ≤ *measure* - *?B*) **using** *pq* **by**(*rule rel-pmf-measureD*)
also have ... ≤ *measure* (*measure-pmf r*) {*z*. ∃ *y*∈*?B*. *S y z*}
using *qr* **by**(*rule rel-pmf-measureD*)
also have {*z*. ∃ *y*∈*?B*. *S y z*} = {*z*. ∃ *x*∈*A*. (*R OO S*) *x z*} **by** *auto*
finally show *measure* (*measure-pmf p*) *A* ≤ *measure* (*measure-pmf r*) ... **.**
qed

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.