

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

Andreas Lochbihler

March 17, 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	3
2	Existence of maximum flows and minimal cuts in finite graphs	8
3	Matrices for given marginals	10
4	Graphs	30
5	Network and Flow	33
5.1	Cut	39
5.2	Countable network	39
5.3	Reduction for avoiding antiparallel edges	40

6	Webs and currents	43
6.1	Saturated and terminal vertices	46
6.2	Separation	47
6.3	Waves	54
6.4	Hindrances and looseness	60
6.5	Linkage	63
6.6	Trimming	64
6.7	Composition of waves via quotients	70
6.8	Well-formed webs	82
6.9	Subtraction of a wave	85
6.10	Bipartite webs	86
7	Reductions	87
7.1	From a web to a bipartite web	87
7.2	Extending a wave by a linkage	105
7.3	From a network to a web	107
7.4	Avoiding antiparallel edges and self-loops	115
7.5	Eliminating zero edges and incoming edges to <i>source</i> and out- going edges of <i>sink</i>	118
8	The max-flow min-cut theorem in bounded networks	119
8.1	Linkages in unhindered bipartite webs	119
8.2	Glueing the reductions together	130
9	Attainability of flows in networks	135
9.1	Cleaning up flows	135
9.2	Residual network	156
9.3	The attainability theorem	158
10	The max-flow min-cut theorems in unbounded networks	186
10.1	More about waves	186
10.2	Hindered webs with reduced weights	192
10.3	Reduced weight in a loose web	210
10.4	Single-vertex saturation in unhindered bipartite webs	238
10.5	Linkability of unhindered bipartite webs	265
10.6	Glueing the reductions together	267
11	The Max-Flow Min-Cut theorem	269
12	Characterisation of <i>rel-pmf</i>	270
12.1	Code generation for <i>rel-pmf</i>	272
13	Characterisation of <i>rel-pmf</i> proved via MFMC	273

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 ( $\lambda x. (x, y)$ ) A
by(simp add: inj-on-def)

lemma inj-map-prod':  $\llbracket \text{inj } f; \text{inj } g \rrbracket \implies \text{inj-on } (\text{map-prod } f \ g) \ X$ 
by(rule subset-inj-on[OF prod.inj-map subset-UNIV])

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

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

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

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

lemma Cauchy-real-Suc-diff:
  fixes  $X :: \text{nat} \Rightarrow \text{real}$  and  $x :: \text{real}$ 
  assumes bounded:  $\bigwedge n. |f (\text{Suc } n) - f \ n| \leq (c / x \wedge n)$ 
  and  $x: 1 < x$ 
  shows Cauchy  $f$ 
proof(cases c > 0)
  case c: True
  show ?thesis
  proof(rule metric-CauchyI)
    fix  $\varepsilon :: \text{real}$ 

```

```

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  $M :: \text{nat}$ 
  where  $\ln (c * x) - \ln (\varepsilon * (x - 1)) < \text{real } M * \ln x$  by blast
  hence  $\exp (\ln (c * x) - \ln (\varepsilon * (x - 1))) < \exp (\text{real } M * \ln x)$  by(rule
exp-less-mono)
  hence  $M: c * (1 / x) ^ M / (1 - 1 / x) < \varepsilon$  using  $\langle 0 < \varepsilon \rangle x c$ 
  by (simp add: exp-diff exp-of-nat-mult field-simps del: ln-mult)

{ fix  $n m :: \text{nat}$ 
  assume  $n \geq M \ m \geq M$ 
  then have  $|f m - f n| \leq c * ((1 / x) ^ M - (1 / x) ^ {\max m n}) / (1 - 1 / x)$ 
  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  $\langle \text{Suc } k = \cdot \rangle$  obtain  $m'$  where  $m: m = \text{Suc } m'$  by(cases m) simp-all
      with  $\langle \text{Suc } k = \cdot \rangle$  Suc.prems have  $k = m' - n \leq m' \ M \leq n \ M \leq m'$ 
by simp-all
      hence  $|f m' - f n| \leq c * ((1 / x) ^ M - (1 / x) ^ {\max m' n}) / (1 - 1 / x)$  by(rule Suc)
      also have  $\dots = c * ((1 / x) ^ M - (1 / x) ^ {m'}) / (1 - 1 / x)$  using
 $\langle n \leq m' \rangle$  by(simp add: max-def)
      moreover
      from bounded have  $|f m - f m'| \leq (c / x ^ {m'})$  by(simp add: m)
      ultimately have  $|f m' - f n| + |f m - f m'| \leq c * ((1 / x) ^ M - (1 / x) ^ {m'}) / (1 - 1 / x) + \dots$  by simp
      also have  $\dots \leq c * ((1 / x) ^ M - (1 / x) ^ m) / (1 - 1 / x)$  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  $\langle n \leq m \rangle$  by(simp add: max-def)
    qed
  qed
  also have  $\dots < c * (1 / x) ^ M / (1 - 1 / x)$  using  $x c$  by(auto simp add: field-simps)
  also have  $\dots < \varepsilon$  using  $M$  .
  finally have  $|f m - f n| < \varepsilon$  . }
  thus  $\exists M. \forall m \geq M. \forall n \geq M. \text{dist } (f m) (f n) < \varepsilon$  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  $\implies$  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:  $\llbracket x \in \text{set } ys; \text{length } xs \geq \text{length } ys \rrbracket \implies \exists z. (z, x) \in \text{set } (\text{zip } xs \text{ } ys)$ 
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  $\longleftrightarrow$  ( $\exists ys' zs'. xs = ys' @ zs' \wedge ys = \text{map } f \text{ } ys' \wedge zs = \text{map } f \text{ } zs'$ )
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)  $\implies x \in \text{set } xs$ 
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*:
 $\text{inj-on } f \ A \implies (\int^+ y. g \ y \ \partial \text{count-space } (f \ ' \ A)) = (\int^+ x. g \ (f \ x) \ \partial \text{count-space } A)$
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* ($\langle (2\sum^+ \text{-} \cdot \text{-} / \text{-}) \rangle$ [*0*,
51, *10*] *10*)

-nn-sum-UNIV :: *pttrn* $\Rightarrow$ '*b* $\Rightarrow$ '*b*::*comm-monoid-add* ($\langle (2\sum^+ \text{-} \cdot \text{-} / \text{-}) \rangle$ [*0*, *10*]
10)

syntax-consts

-nn-sum -nn-sum-UNIV $\equiv$ *nn-integral*

translations

$\sum^+_{i \in A}. b \equiv \text{CONST } \text{nn-integral } (\text{CONST } \text{count-space } A) \ (\lambda i. b)$
 $\sum^+ i. b \equiv \sum^+_{i \in \text{CONST UNIV}}. b$

inductive-simps *rtrancl-path-simps*:

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

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

lemma *in-restrict-rel-iff*: $(x, y) \in \text{restrict-rel } A \ R \longleftrightarrow (x, y) \in R \wedge x \in A \wedge y \in A$
by(*simp* *add*: *restrict-rel-def*)

lemma *restrict-relE*: $\llbracket (x, y) \in \text{restrict-rel } A \ R; \llbracket (x, y) \in R; x \in A; y \in A \rrbracket \implies$
thesis $\rrbracket \implies$ *thesis*
by(*simp* *add*: *restrict-rel-def*)

lemma *restrict-relI* [*intro!*]: $\llbracket (x, y) \in R; x \in A; y \in A \rrbracket \implies (x, y) \in \text{restrict-rel}$
A *R*
by(*simp* *add*: *restrict-rel-def*)

lemma *Field-restrict-rel-subset*: *Field* (*restrict-rel* *A* *R*) $\subseteq A \cap \text{Field } R$
by(*auto* *simp* *add*: *Field-def in-restrict-rel-iff*)

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


```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

lemma incseq-chain-range:  $\text{incseq } f \implies \text{Complete-Partial-Order.chain } (\leq) (\text{range } f)$ 
apply ( $\text{rule chainI; clarsimp}$ )
    using linear by ( $\text{auto dest: incseqD}$ )

```

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

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

end

context *NFlow* **begin**

lemma *conservation*: $\llbracket x \neq s; x \neq t \rrbracket \implies \text{sum } f (\text{incoming } x) = \text{sum } f (\text{outgoing } x)$
by(*cases x ∈ V*)(*auto simp add: conservation-const outgoing-outside incoming-outside*)

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

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

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

end

context *Network* **begin**

definition *shortest-augmenting-path* $f = (\text{SOME } p. \text{Graph.isShortestPath } (\text{residualGraph } c \ f) \ s \ p \ t)$

lemma *shortest-augmenting-path*:
assumes *NFlow c s t f*
and $\exists p. \text{NPreflow.isAugmentingPath } c \ s \ t \ f \ p$
shows $\text{Graph.isShortestPath } (\text{residualGraph } c \ f) \ s \ (\text{shortest-augmenting-path } f) \ t$
using *assms* **unfolding** *shortest-augmenting-path-def*
by *clarify*(*rule someI-ex, rule NFlow.augmenting-path-imp-shortest*)

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

lemma *max-flow*:
 $\text{NFlow } c \ s \ t \ \text{max-flow} \ (\text{is ?thesis1})$

```

  ¬ (∃ 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 short-
est-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 (λf. 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: ⋀X. X ⊆ {..n} ⇒ sum f X ≤ sum g (R “ X)
    and f-nonneg: ⋀x. 0 ≤ f x
    and g-nonneg: ⋀y. 0 ≤ g y
    and R: R ⊆ {..n} × {..n}
  obtains h :: nat ⇒ nat ⇒ real

```

where $\bigwedge x y. \llbracket x \leq n; y \leq n \rrbracket \implies 0 \leq h x y$
 and $\bigwedge x y. \llbracket 0 < h x y; x \leq n; y \leq n \rrbracket \implies (x, y) \in R$
 and $\bigwedge x. x \leq n \implies f x = \text{sum } (h x) \{..n\}$
 and $\bigwedge y. y \leq n \implies g y = \text{sum } (\lambda x. h x y) \{..n\}$
proof(cases $\exists x \leq n. f x > 0$)
 case *False*
 hence $f x = 0$ **if** $x \leq n$ **for** x **using** $f\text{-nonneg}[of x]$ **that** **by**(auto simp add: not-less)
 hence $\text{sum } g \{..n\} = 0$ **using** eq-sum **by** simp
 hence $g y = 0$ **if** $y \leq n$ **for** y **using** $g\text{-nonneg}$ **that** **by**(simp add: sum-nonneg-eq-0-iff)
 with f **show** thesis **by**(auto intro!: that[of $\lambda -. 0$])
 next
 case *True*
 then obtain $x0$ **where** $x0: x0 \leq n \wedge f x0 > 0$ **by** blast

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

 have [simp]: finite ($R \restriction A$) **for** A
 proof –
 have $R \restriction A \subseteq \{..n\}$ **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: $R'\text{-def}$)
 have R'' : $R' \restriction A \subseteq \{..n\}$ **for** A **using** R **by**(auto simp add: $R'\text{-def}$)
 have [simp]: finite ($R' \restriction A$) **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** $x0: x0 \leq n \wedge f x0 > 0$ **for** $x0$
 proof(rule ccontr)
 assume $\neg$?thesis
 then have $g y0 = 0$ **if** $(x0, y0) \in R$ $y0 \leq n$ **for** $y0$ **using** $g\text{-nonneg}[of y0]$
 that **by** auto
 moreover from R **have** $R \restriction \{x0\} \subseteq \{..n\}$ **by** auto
 ultimately have $\text{sum } g (R \restriction \{x0\}) = 0$
 using $g\text{-nonneg}$ **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** $R \restriction \{x0\} \neq \{\}$ $\text{sum } g (R \restriction \{x0\}) > 0$ **by** auto
 ultimately show False **by** simp
 qed
 then obtain $y0$ **where** $y0: y0 \leq n \wedge (x0, y0) \in R' \wedge g y0 > 0$ **using** $x0$ **by**(auto simp add: $R'\text{-def}$)

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

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

```

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

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

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

```

```

    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:  $\llbracket v' \leq n; (v', v - n - 1) \in R \rrbracket \implies f v' = 0$  for v' using
f-nonneg[of v'] by auto
    have sum f {..n} = sum f {x. x  $\leq$  n  $\wedge$  x  $\notin$  R-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-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  $\langle s \in V \rangle$  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 =  $\lambda x y. \text{max-flow } (x, y + n + 1)$ 
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 ( $\lambda(-, x). f \ x$ ) (Pair s ‘  $\{..n\}$ )
  by(rule sum.mono-neutral-cong-left)(auto simp add: outgoing-def E-def)
also have ... = 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: ( $\lambda y. y + n + 1$ ) ‘ ( $R' \text{ `` } (C \cap \{..n\})$ )  $\subseteq C$ 
proof(rule ccontr)
  assume  $\neg ?thesis$ 
  then obtain x y where *: ( $x, y$ )  $\in R' \ x \leq n \ x \in C \ y + n + 1 \notin C$  by auto
  then have xy: ( $x, y + n + 1$ )  $\in E \ y \leq n \ c \ (x, y + n + 1) = LARGE$ 
    using R by(auto simp add: E-def c-simps R'-def)

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

have  $C.cap \leq \text{sum } f \ \{..n\}$  using out h.val-bounded(2) eq by simp
then have cap:  $C.cap = \text{sum } f \ \{..n\}$ 
proof(rule antisym)
  let ?L =  $\{x. x \leq n \wedge x \in C \wedge f \ x > 0\}$ 
  let ?R = ( $\lambda y. y + n + 1$ ) ‘ ( $R' \text{ `` } ?L$ )
  have sum f  $\{..n\} = \text{sum } f \ ?L + \text{sum } f \ (\{..n\} - ?L)$  by(subst sum-diff) auto
  also have sum f ?L  $\leq \text{sum } g \ (R \text{ `` } ?L)$  by(rule le) auto
  also have ... = 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 ... = sum c (( $\lambda y. (y + n + 1, t)$ ) ‘ ( $R' \text{ `` } ?L$ )) 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 ‘ ( $\{..n\} - ?L$ )) by(simp add:
sum.reindex inj-on-def)
  also have sum c (( $\lambda y. (y + n + 1, t)$ ) ‘ ( $R' \text{ `` } ?L$ )) + ... =
    sum c ((( $\lambda y. (y + n + 1, t)$ ) ‘ ( $R' \text{ `` } ?L$ ))  $\cup$  (Pair s ‘ ( $\{..n\} - ?L$ )))

```

```

    by(subst sum.union-disjoint) auto
    also have ... ≤ sum c (((λy. (y + n + 1, t)) ‘ (R’ “ ?L)) ∪ (Pair s ‘ ({..n}
- ?L)) ∪ {(x, t) | x. x ∈ C ∧ n < x ∧ x ≤ 2 * n + 1})
    by(rule sum-mono2)(auto simp add: g-nonneg)
    also have (((λy. (y + n + 1, t)) ‘ (R’ “ ?L)) ∪ (Pair s ‘ ({..n} - ?L)) ∪ {(x,
t) | x. x ∈ C ∧ n < x ∧ x ≤ 2 * n + 1}) = (Pair s ‘ ({..n} - ?L)) ∪ {(x, t) |
x. x ∈ C ∧ n < x ∧ x ≤ 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} ≤ ... by simp
qed

show thesis
proof
  show 0 ≤ ?h x y for x y by(rule h.f-non-negative)
  show (x, y) ∈ R if 0 < ?h x y x ≤ n y ≤ 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 ≤ 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} ≤ f x if x ≤ n for x
    using sum-h[OF that] h.capacity-const[rule-format, of (s, x)] that by simp
  moreover have f x ≤ sum (?h x) {..n} if x ≤ n for x
  proof(rule ccontr)
    assume ¬ ?thesis
    hence sum (?h x) {..n} < f x by simp
    hence sum (λx. (sum (?h x) {..n})) {..n} < sum f {..n}

```

```

    using le that by(intro sum-strict-mono-ex1) auto
  also have sum (λx. (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 ≤ n for x using that by(auto
intro: antisym)

have sum-h': sum (λx. ?h x y) {..n} = max-flow (y + n + 1, t) if y ≤ n for y
proof -
  have sum (λx. ?h x y) {..n} = sum max-flow ((λx. (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 (λx. ?h x y) {..n} ≤ g y if y ≤ 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 ≤ sum (λx. ?h x y) {..n} if y ≤ n for y
proof(rule ccontr)
  assume ¬ ?thesis
  hence sum (λx. ?h x y) {..n} < g y by simp
  hence sum (λy. (sum (λx. ?h x y) {..n})) {..n} < sum g {..n}
    using le' that by(intro sum-strict-mono-ex1) auto
  also have sum (λy. (sum (λx. ?h x y) {..n})) {..n} = sum max-flow ((λy. (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  $g\ y = \text{sum } (\lambda x. ?h\ x\ y)\ \{..n\}$  if  $y \leq n$  for  $y$  using that
  by(auto intro: antisym)
  qed
qed

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

lemma convergent-bounded-family:
  fixes  $f :: \text{nat} \Rightarrow 'a \Rightarrow \text{real}$ 
  assumes bounded:  $\bigwedge x. x \in A \implies \text{bounded } (\text{range } (\lambda n. f\ n\ x))$ 
  and  $A$ : countable  $A$ 
  obtains  $k$  where strict-mono  $k \bigwedge x. x \in A \implies \text{convergent } (\lambda n. f\ (k\ n)\ x)$ 
proof(cases  $A = \{\}$ )
  case False
    define  $f'$  where  $f'\ n\ x = f\ n\ (\text{from-nat-into } A\ x)$  for  $n\ x$ 
    have bounded  $(\text{range } (\lambda n. f'\ n\ x))$  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  $(\lambda n. f'\ (k\ n)\ x)$  for  $x$ 
      by(rule convergent-bounded-family-nat) iprover
    have convergent  $(\lambda n. f\ (k\ n)\ x)$  if  $x \in A$  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 $\text{zero-on} :: ('a \Rightarrow 'b :: \text{zero}) \Rightarrow 'a \text{ set} \Rightarrow 'a \Rightarrow 'b$
where $\text{zero-on } f \equiv \text{override-on } f (\lambda \cdot. 0)$

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

lemma zero-on-nonneg : **fixes** $f :: 'a \Rightarrow 'b :: \{\text{preorder}, \text{zero}\}$ **shows**
 $0 \leq \text{zero-on } f \ X \ x \longleftrightarrow (x \notin X \longrightarrow 0 \leq f \ x)$
by(*auto simp add: override-on-def*)

lemma sums-zero-on :
fixes $f :: \text{nat} \Rightarrow 'a :: \text{real-normed-vector}$
assumes $f: f \text{ sums } s$
and $X: \text{finite } X$
shows $\text{zero-on } f \ X \text{ sums } (s - \text{sum } f \ X)$
proof –
have $(\lambda x. \text{if } x \in X \text{ then } f \ x \text{ else } 0) \text{ sums } \text{sum } (\lambda x. f \ x) \ X$ **using** X **by**(*rule sums-If-finite-set*)
from $\text{sums-diff}[OF \ f \ \text{this}]$ **show** $?thesis$
by(*simp add: sum-negf override-on-def if-distrib cong del: if-weak-cong*)
qed

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

lemma $\text{summable-zero-on-nonneg}$:
fixes $f :: \text{nat} \Rightarrow 'a :: \{\text{ordered-comm-monoid-add}, \text{linorder-topology}, \text{conditionally-complete-linorder}\}$
assumes $f: \text{summable } f$
and $\text{nonneg}: \bigwedge x. 0 \leq f \ x$
shows $\text{summable } (\text{zero-on } f \ X)$
proof(*rule summableI-nonneg-bounded*)
fix n
have $\text{sum } (\text{zero-on } f \ X) \ \{..<n\} \leq \text{sum } f \ \{..<n\}$ **by**(*rule sum-mono*)(*simp add: nonneg*)
also have $\dots \leq \text{suminf } f$ **using** f **by**(*rule sum-le-suminf*)(*auto simp add: nonneg*)
finally show $\text{sum } (\text{zero-on } f \ X) \ \{..<n\} \leq \text{suminf } f$.
qed(*simp add: zero-on-nonneg nonneg*)

lemma *zero-on-ennreal* [*simp*]: $\text{zero-on } (\lambda x. \text{ennreal } (f x)) A = (\lambda x. \text{ennreal } (\text{zero-on } f A x))$

by(*simp add: override-on-def fun-eq-iff*)

lemma *sum-lessThan-conv-atMost-nat*:

fixes $f :: \text{nat} \Rightarrow 'b :: \text{ab-group-add}$

shows $\text{sum } f \{..<n\} = \text{sum } f \{..n\} - f n$

by (*metis Groups.add-ac(2) add-diff-cancel-left' lessThan-Suc-atMost sum.lessThan-Suc*)

lemma *Collect-disjoint-atLeast*:

$\text{Collect } P \cap \{x.. \} = \{ \} \longleftrightarrow (\forall y \geq x. \neg P y)$

by(*auto simp add: atLeast-def*)

lemma *bounded-matrix-for-marginals-nat*:

fixes $f g :: \text{nat} \Rightarrow \text{real}$

and $R :: (\text{nat} \times \text{nat}) \text{ set}$

and $s :: \text{real}$

assumes *sum-f*: $f \text{ sums } s$ **and** *sum-g*: $g \text{ sums } s$

and *f-nonneg*: $\bigwedge x. 0 \leq f x$ **and** *g-nonneg*: $\bigwedge y. 0 \leq g y$

and *f-le-g*: $\bigwedge X. \text{suminf } (\text{zero-on } f (- X)) \leq \text{suminf } (\text{zero-on } g (- R \text{ `` } X))$

obtains $h :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{real}$

where $\bigwedge x y. 0 \leq h x y$

and $\bigwedge x y. 0 < h x y \implies (x, y) \in R$

and $\bigwedge x. h x \text{ sums } f x$

and $\bigwedge y. (\lambda x. h x y) \text{ sums } g y$

proof –

have *summ-f*: *summable* f **and** *summ-g*: *summable* g **and** *sum-fg*: $\text{suminf } f = \text{suminf } g$

using *sum-f sum-g* **by**(*auto simp add: summable-def sums-unique[symmetric]*)

have *summ-zf*: *summable* $(\text{zero-on } f A)$ **for** A

using *summ-f f-nonneg* **by**(*rule summable-zero-on-nonneg*)

have *summ-zg*: *summable* $(\text{zero-on } g A)$ **for** A

using *summ-g g-nonneg* **by**(*rule summable-zero-on-nonneg*)

define $f' :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{real}$

where $f' n x = (\text{if } x \leq n \text{ then } f x \text{ else if } x = \text{Suc } n \text{ then } \sum k. f (k + (n + 1)) \text{ else } 0)$ **for** $n x$

define $g' :: \text{nat} \Rightarrow \text{nat} \Rightarrow \text{real}$

where $g' n y = (\text{if } y \leq n \text{ then } g y \text{ else if } y = \text{Suc } n \text{ then } \sum k. g (k + (n + 1)) \text{ else } 0)$ **for** $n y$

define $R' :: \text{nat} \Rightarrow (\text{nat} \times \text{nat}) \text{ set}$

where $R' n =$

$R \cap \{..n\} \times \{..n\} \cup$

$\{(n + 1, y) \mid y. \exists x' > n. (x', y) \in R \wedge y \leq n\} \cup$

$\{(x, n + 1) \mid x. \exists y' > n. (x, y') \in R \wedge x \leq n\} \cup$

$(\text{if } \exists x > n. \exists y > n. (x, y) \in R \text{ then } \{(n + 1, n + 1)\} \text{ else } \{ \})$

for n

have *R'-sims* [*simplified*, *simp*]:

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

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

have R' -intros:

$\llbracket (x, y) \in R; x \leq n; y \leq n \rrbracket \implies (x, y) \in R' n$
 $\llbracket (x', y) \in R; n < x'; y \leq n \rrbracket \implies (n + 1, y) \in R' n$
 $\llbracket (x, y') \in R; x \leq n; n < y' \rrbracket \implies (x, n + 1) \in R' n$
 $\llbracket (x', y') \in R; n < x'; n < y' \rrbracket \implies (n + 1, n + 1) \in R' n$
for $n x y x' y'$ **by**(*auto*)

have $Image\text{-}R'$:

$R' n \text{ `` } X = (R \text{ `` } (X \cap \{..n\})) \cap \{..n\} \cup$
 $(\text{if } n + 1 \in X \text{ then } (R \text{ `` } \{n+1..\}) \cap \{..n\} \text{ else } \{\}) \cup$
 $(\text{if } (R \text{ `` } (X \cap \{..n\})) \cap \{n+1..\} = \{\} \text{ then } \{\} \text{ else } \{n + 1\}) \cup$
 $(\text{if } n + 1 \in X \wedge (R \text{ `` } \{n+1..\}) \cap \{n+1..\} \neq \{\} \text{ then } \{n + 1\} \text{ else } \{\})$ **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 $\text{sum } (f' n) \{..n + 1\} = \text{sum } (g' n) \{..n + 1\}$ **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 atLeast0AtMost atLeast0LessThan atLeastLessThanSuc-atLeastAtMost minus-add-distrib sum.lessThan-Suc uminus-add-conv-diff*)

moreover have $\text{sum } (f' n) X \leq \text{sum } (g' n) (R' n \text{ `` } X)$ **if** $X \subseteq \{..n + 1\}$ **for**

X

proof –

from *that* **have** [*simp*]: *finite X* **by**(*rule finite-subset*) *simp*

define X' **where** $X' \equiv \text{if } n + 1 \in X \text{ then } X \cup \{n+1..\} \text{ else } X$

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

```

    have sum (f' n) X = sum (f' n) (X - {n + 1}) + (if n + 1 ∈ 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 ... + (if n + 1 ∈ X then f' n (n + 1) else 0) = suminf (zero-on
f (- X'))
    proof(cases n + 1 ∈ 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 ... ≤ suminf (zero-on g (- R “ X')) by(rule f-le-g)
also have ... ≤ suminf (zero-on g (- Y'))
    by(rule suminf-le[OF - summ-zg summ-zg])(clarsimp simp add: over-
ride-on-def g-nonneg Y'-def summ-zg)
also have ... = suminf (λk. zero-on g (- Y') (k + (n + 1))) + sum (zero-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-on g (- Y')) {..n} = sum g (Y' ∩ {..n})
    by(rule sum.mono-neutral-cong-right)(auto simp add: override-on-def)
also have ... = sum (g' n) (Y' ∩ {..n})
    by(rule sum.cong)(auto simp add: g'-def)
also have suminf (λk. zero-on g (- Y') (k + (n + 1))) ≤ (if R “ X' ∩
{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 ... + sum (g' n) (Y' ∩ {..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 ∧x. 0 ≤ f' n x ∧y. 0 ≤ 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  $\bigwedge x y. \llbracket x \leq n+1; y \leq n+1 \rrbracket \implies 0 \leq h x y$ 
  and  $\bigwedge x y. \llbracket 0 < h x y; x \leq n+1; y \leq n+1 \rrbracket \implies (x, y) \in R' n$ 
  and  $\bigwedge x. x \leq n+1 \implies f' n x = \text{sum } (h x) \{..n+1\}$ 
  and  $\bigwedge y. y \leq n+1 \implies g' n y = \text{sum } (\lambda x. h x y) \{..n+1\}$ 
  by(rule bounded-matrix-for-marginals-finite) blast+
  hence  $\exists h. (\forall x y. x \leq n+1 \longrightarrow y \leq n+1 \longrightarrow 0 \leq h x y) \wedge$ 
     $(\forall x y. 0 < h x y \longrightarrow x \leq n+1 \longrightarrow y \leq n+1 \longrightarrow (x, y) \in R' n) \wedge$ 
     $(\forall x. x \leq n+1 \longrightarrow f' n x = \text{sum } (h x) \{..n+1\}) \wedge$ 
     $(\forall y. y \leq n+1 \longrightarrow g' n y = \text{sum } (\lambda x. h x y) \{..n+1\})$  by blast }
  hence  $\exists h. \forall n. (\forall x y. x \leq n+1 \longrightarrow y \leq n+1 \longrightarrow 0 \leq h n x y) \wedge$ 
     $(\forall x y. 0 < h n x y \longrightarrow x \leq n+1 \longrightarrow y \leq n+1 \longrightarrow (x, y) \in R' n) \wedge$ 
     $(\forall x. x \leq n+1 \longrightarrow f' n x = \text{sum } (h n x) \{..n+1\}) \wedge$ 
     $(\forall y. y \leq n+1 \longrightarrow g' n y = \text{sum } (\lambda x. h n x y) \{..n+1\})$ 
  by(subst choice-iff[symmetric]) blast
  then obtain h where h-nonneg:  $\bigwedge n x y. \llbracket x \leq n+1; y \leq n+1 \rrbracket \implies 0 \leq h$ 
 $n x y$ 
  and h-R:  $\bigwedge n x y. \llbracket 0 < h n x y; x \leq n+1; y \leq n+1 \rrbracket \implies (x, y) \in R' n$ 
  and h-f:  $\bigwedge n x. x \leq n+1 \implies f' n x = \text{sum } (h n x) \{..n+1\}$ 
  and h-g:  $\bigwedge n y. y \leq n+1 \implies g' n y = \text{sum } (\lambda x. h n x y) \{..n+1\}$ 
  apply(rule exE)
  subgoal for h by(erule meta-allE[of - h]) blast
  done

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

have  $h' n xy \leq s$  for n xy
proof(cases xy)
  case [simp]: (Pair x y)
  consider (le)  $x \leq n \wedge y \leq n \mid (\text{beyond}) x > n \vee y > n$  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  $\dots \leq h n x y + \text{sum } (h n x) \{..<y\} + \text{sum } (h n x) \{y<..n+1\}$ 
    using h-nonneg le by(auto intro!: sum-nonneg add-nonneg-nonneg)
    also have  $\dots = \text{sum } (h n x) \{..y\} + \text{sum } (h n x) \{y<..n+1\}$ 
    by(simp add: sum-lessThan-conv-atMost-nat)
    also have  $\dots = \text{sum } (h n x) \{..n+1\}$  using le
    by(subst sum.union-disjoint[symmetric])(auto simp del: One-nat-def intro!:
sum.cong)
    also have  $\dots = f' n x$  using le by(simp add: h-f)
    also have  $\dots = f x$  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 ...  $\leq$  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  $0 \leq s$  using summ-f by(simp add: sums-unique[OF sum-f] sum-
inf-nonneg f-nonneg)
  finally show ?thesis .
qed
qed
then have bounded (range ( $\lambda n. h' n x$ )) 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:  $\bigwedge xy. \text{convergent } (\lambda n. h' (k n) xy)$  by auto

define H ::  $\text{nat} \Rightarrow \text{nat} \Rightarrow \text{real}$ 
  where  $H x y = \lim (\lambda n. h' (k n) (x, y))$  for x y

have H:  $(\lambda n. h' (k n) (x, y)) \longrightarrow H x y$  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:  $0 \leq H x y$  for x y using H[of x y] by(rule LIMSEQ-le-const)(simp
add: h'-nonneg)
  show  $(x, y) \in R$  if  $0 < H x y$  for x y
  proof(rule ccontr)
    assume  $(x, y) \notin R$ 
    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  $H x \text{ sums } f x$  for x unfolding sums-iff
proof
  have sum-H:  $\text{sum } (H x) \{.. $m\} \leq f x$  for m
  proof -
    have  $\text{sum } (\lambda y. h' (k n) (x, y)) \{.. $m\} \leq f x$  for n
    proof(cases  $x \leq k n$ )
      case True
        from k have  $n \leq k n$  by(rule seq-suble)
        have  $\text{sum } (\lambda y. h' (k n) (x, y)) \{.. $m\} = \text{sum } (\lambda 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 ...  $\leq \text{sum } (\lambda 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) ≤ f x using sum-H by(rule suminf-le-const)
moreover
  have (λm. sum (H x) {..

```



```

      subgoal by auto
    done
    finally show  $f x \leq \text{sum } (\lambda y. h' (k n) (x, y)) \{..<m\} + \dots$  by simp
  qed
  ultimately show ?thesis by (rule LIMSEQ-le-const)
  qed
  thus  $\exists N. \forall n \geq N. f x \leq \text{sum } (H x) \{..<n\} + (\sum i. g (i + n))$  by auto
  qed
  ultimately show  $\text{suminf } (H x) = f x$  by simp
  qed
  show  $(\lambda x. H x y)$  sums  $g y$  for  $y$  unfolding sums-iff
  proof
    have  $\text{sum-H: sum } (\lambda x. H x y) \{..<m\} \leq g y$  for  $m$ 
    proof -
      have  $\text{sum } (\lambda x. h' (k n) (x, y)) \{..<m\} \leq g y$  for  $n$ 
      proof (cases  $y \leq k n$ )
        case True
        from  $k$  have  $n \leq k n$  by (rule seq-suble)
        have  $\text{sum } (\lambda x. h' (k n) (x, y)) \{..<m\} = \text{sum } (\lambda x. h' (k n) (x, y)) \{..<\min$ 
 $m (k n + 1)\}$ 
        by (rule sum.mono-neutral-right) (auto simp add: h'-def min-def)
        also have  $\dots \leq \text{sum } (\lambda x. h (k n) x y) \{..k n + 1\}$  using True
        by (intro sum-le-included [where  $i=id$ ]) (auto simp add: h'-def h-nonneg)
        also have  $\dots = g' (k n) y$  using h-g True by simp
        also have  $\dots = g y$  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  $\text{summ-H: summable } (\lambda x. H x y)$  by (rule summableI-nonneg-bounded)
  hence  $\text{suminf } (\lambda x. H x y) \leq g y$  using sum-H by (rule suminf-le-const)
  moreover
  have  $(\lambda m. \text{sum } (\lambda x. H x y) \{..<m\} + \text{suminf } (\lambda n. f (n + m))) \longrightarrow \text{suminf}$ 
 $(\lambda x. H x y) + 0$ 
  by (rule tendsto-intros summable-LIMSEQ summ-H suminf-exist-split2 summ-f)+
  hence  $g y \leq \text{suminf } (\lambda x. H x y) + 0$ 
  proof (rule LIMSEQ-le-const)
    have  $g y \leq \text{sum } (\lambda x. H x y) \{..<m\} + \text{suminf } (\lambda n. f (n + m))$  for  $m$ 
    proof -
      have  $(\lambda n. \text{sum } (\lambda x. h' (k n) (x, y)) \{..<m\} + \text{suminf } (\lambda i. f (i + m)))$ 
 $\longrightarrow \text{sum } (\lambda x. H x y) \{..<m\} + \text{suminf } (\lambda i. f (i + m))$ 
      by (rule tendsto-intros H)+
      moreover have  $\exists N. \forall n \geq N. g y \leq \text{sum } (\lambda x. h' (k n) (x, y)) \{..<m\} +$ 
 $\text{suminf } (\lambda i. f (i + m))$ 
      proof (intro exI strip)
        fix  $n$ 
        assume  $\max y m \leq n$ 
        with seq-suble [OF  $k$ , of  $n$ ] have  $y: y \leq k n$  and  $m: m \leq k n$  by auto
        have  $g y = g' (k n) y$  using  $y$  by (simp add: g'-def)

```

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

lemma *bounded-matrix-for-marginals-ennreal*:

assumes *sum-eq*: $(\sum^+ x \in A. f x) = (\sum^+ y \in B. g y)$
and *finite*: $(\sum^+ x \in B. g x) \neq \top$
and *le*: $\bigwedge X. X \subseteq A \implies (\sum^+ x \in X. f x) \leq (\sum^+ y \in R. X. g y)$
and *countable* [*simp*]: *countable A countable B*
and *R*: $R \subseteq A \times B$
obtains h **where** $\bigwedge x y. 0 < h x y \implies (x, y) \in R$
and $\bigwedge x y. h x y \neq \top$
and $\bigwedge x. x \in A \implies (\sum^+ y \in B. h x y) = f x$
and $\bigwedge y. y \in B \implies (\sum^+ x \in A. h x y) = g y$
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  $f'\ x = (\text{if } x \in \text{to-nat-on } A \text{ ' } A \text{ then } \text{enn2real } (f \text{ (from-nat-into } A\ x)) \text{ else } 0)$  for  $x$ 
define g' where  $g'\ y = (\text{if } y \in \text{to-nat-on } B \text{ ' } B \text{ then } \text{enn2real } (g \text{ (from-nat-into } B\ y)) \text{ else } 0)$  for  $y$ 
define s where  $s = \text{enn2real } (\sum^+ x \in B. g\ x)$ 
define R' where  $R' = \text{map-prod } (\text{to-nat-on } A) (\text{to-nat-on } B) \text{ ' } R$ 

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  $(\sum^+ x. f'\ x) = (\sum^+ x \in \text{to-nat-on } A \text{ ' } A. f'\ x)$ 
  by(auto simp add: nn-integral-count-space-indicator f'-def intro!: nn-integral-cong)
also have  $\dots = (\sum^+ x \in A. f\ x)$ 
  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  $(\sum^+ y. g'\ y) = (\sum^+ y \in \text{to-nat-on } B \text{ ' } B. g'\ y)$ 
  by(auto simp add: nn-integral-count-space-indicator g'-def intro!: nn-integral-cong)
also have  $\dots = (\sum^+ y \in B. g\ y)$ 
  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  $\text{sum } f' \{.. for  $n$ 
  proof -
    have  $\text{sum } f' \{..
    by(simp add: nn-integral-count-space-finite f'-nonneg sum-nonneg)
    also have  $\text{enn2real } (\sum^+ x \in \{.. 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':  $\text{suminf } f' = \text{enn2real } (\sum^+ y. f'\ y)$ 
  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  $\text{sum } g' \{..<n\} \leq \text{enn2real } (\sum^+ y. g' y)$  for  $n$ 
proof –
  have  $\text{sum } g' \{..<n\} = \text{enn2real } (\sum^+ y \in \{..<n\}. g' y)$ 
  by(simp add: nn-integral-count-space-finite g'-nonneg sum-nonneg)
  also have  $\text{enn2real } (\sum^+ y \in \{..<n\}. g' y) \leq \text{enn2real } (\sum^+ y. g' y)$  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  $\text{suminf-}g': \text{suminf } g' = \text{enn2real } (\sum^+ y. g' y)$ 
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  $\text{suminf } (\text{zero-on } f' (- X)) \leq \text{suminf } (\text{zero-on } g' (- R' \text{ `` } X))$ 
for  $X$ 
proof –
  define  $X'$  where  $X' = \text{from-nat-into } A \text{ `` } (X \cap \text{to-nat-on } A \text{ `` } A)$ 
have  $X': \text{to-nat-on } A \text{ `` } X' = X \cap (\text{to-nat-on } A \text{ `` } A)$ 
by(auto 4 3 simp add: X'-def intro: rev-image-eqI)

  have  $\text{ennreal } (\text{suminf } (\text{zero-on } f' (- X))) = \text{suminf } (\text{zero-on } (\lambda x. \text{ennreal } (f' x)) (- X))$ 
by(simp add: suminf-ennreal2 zero-on-nonneg f'-nonneg summable-zero-on-nonneg summ-f')
  also have  $\dots = (\sum^+ x \in X. f' x)$ 
by(auto simp add: nn-integral-count-space-nat[symmetric] nn-integral-count-space-indicator
intro!: nn-integral-cong split: split-indicator)
  also have  $\dots = (\sum^+ x \in \text{to-nat-on } A \text{ `` } X'. f' x)$  using  $X'$ 
by(auto simp add: nn-integral-count-space-indicator f'-def intro!: nn-integral-cong
split: split-indicator)
  also have  $\dots = (\sum^+ x \in X'. f x)$ 
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  $\dots \leq (\sum^+ y \in R \text{ `` } X'. g y)$  by(rule le)(auto simp add: X'-def)
  also have  $\dots = (\sum^+ y \in \text{to-nat-on } B \text{ `` } (R \text{ `` } X'). g' y)$  using  $R \text{ fin-g}$ 
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  $\text{to-nat-on } B \text{ `` } (R \text{ `` } X') = R' \text{ `` } X$  using  $R$ 
by(auto 4 4 simp add: X'-def R'-def Image-iff intro: rev-image-eqI rev-bexI
intro!: imageI)
  also have  $(\sum^+ y \in \dots. g' y) = \text{suminf } (\text{zero-on } (\lambda y. \text{ennreal } (g' y)) (- \dots))$ 
by(auto simp add: nn-integral-count-space-nat[symmetric] nn-integral-count-space-indicator
intro!: nn-integral-cong split: split-indicator)
  also have  $\dots = \text{ennreal } (\text{suminf } (\text{zero-on } g' (- R' \text{ `` } X)))$ 
by(simp add: suminf-ennreal2 zero-on-nonneg g'-nonneg summable-zero-on-nonneg)

```

```

summ-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:  $\bigwedge x y. 0 \leq h' x y$ 
  and dom-h':  $\bigwedge x y. 0 < h' x y \implies (x, y) \in R'$ 
  and h'-f:  $\bigwedge x. h' x \text{ sums } f' x$ 
  and h'-g:  $\bigwedge y. (\lambda x. h' x y) \text{ sums } g' y$ 
  by(rule bounded-matrix-for-marginals-nat) blast

define h where h x y = ennreal (if x ∈ A ∧ y ∈ B then h' (to-nat-on A x)
(to-nat-on B y) else 0) for x y
show ?thesis
proof
  show (x, y) ∈ 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 ≠ ⊤ for x y by(simp add: h-def)

  fix x
  assume x: x ∈ A
  have  $(\sum^+ y \in B. h x y) = (\sum^+ y \in \text{to-nat-on } B \text{ ' } B. h' (\text{to-nat-on } A x) y)$ 
    by(subst nn-integral-count-space-reindex)(auto simp add: inj-on-to-nat-on h-def
x intro!: nn-integral-cong)
  also have  $\dots = (\sum^+ y. h' (\text{to-nat-on } A x) y)$  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  $\dots = \text{ennreal} (\text{suminf } (h' (\text{to-nat-on } A x)))$ 
  by(simp add: nn-integral-count-space-nat suminf-ennreal-eq[OF h'-f] h'-nonneg)

  also have  $\dots = \text{ennreal} (f' (\text{to-nat-on } A x))$  using h'-f[of to-nat-on A x]
by(simp add: sums-iff)
  also have  $\dots = f x$  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 ∈ B
  have  $(\sum^+ x \in A. h x y) = (\sum^+ x \in \text{to-nat-on } A \text{ ' } A. h' x (\text{to-nat-on } B y))$ 
    by(subst nn-integral-count-space-reindex)(auto simp add: inj-on-to-nat-on h-def
y intro!: nn-integral-cong)
  also have  $\dots = (\sum^+ x. h' x (\text{to-nat-on } B y))$  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  $\dots = \text{ennreal} (\text{suminf } (\lambda x. h' x (\text{to-nat-on } B y)))$ 
  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 ( $\langle \mathbf{E}_1 \rangle$ )
where  $\mathbf{E}_G \equiv \{(x, y). \text{edge } G \ x \ y\}$ 

```

```

definition outgoing :: ('v, 'more) graph-scheme  $\Rightarrow$  'v  $\Rightarrow$  'v set ( $\langle \mathbf{OUT}_1 \rangle$ )
where  $\mathbf{OUT}_G \ x = \{y. (x, y) \in \mathbf{E}_G\}$ 

```

```

definition incoming :: ('v, 'more) graph-scheme  $\Rightarrow$  'v  $\Rightarrow$  'v set ( $\langle \mathbf{IN}_1 \rangle$ )
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  $\Gamma \ x \ y \Longrightarrow$  vertex  $\Gamma \ x$ 
  and vertexI2: edge  $\Gamma \ x \ y \Longrightarrow$  vertex  $\Gamma \ y$ 
by(auto simp add: vertex-def)

```

```

abbreviation vertices :: ('v, 'more) graph-scheme  $\Rightarrow$  'v set ( $\langle \mathbf{V}_1 \rangle$ )
where  $\mathbf{V}_G \equiv \text{Collect } (\text{vertex } G)$ 

```

```

lemma V-def:  $\mathbf{V}_G = \text{fst } \text{' } \mathbf{E}_G \cup \text{snd } \text{' } \mathbf{E}_G$ 

```

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* $\Rightarrow$ *'v* $\Rightarrow$ *'v path* $\Rightarrow$ *'v* $\Rightarrow$ *bool*
where *path* *G* $\equiv$ *rtrancl-path* (*edge* *G*)

inductive *cycle* :: (*'v, 'more*) *graph-scheme* $\Rightarrow$ *'v path* $\Rightarrow$ *bool*
for *G*

where — Cycles must not pass through the same node multiple times. Otherwise, the cycle might enter a node via two different edges and leave it via just one edge. Thus, the clean-up lemma would not hold any more.

cycle: $\llbracket \text{path } G \ v \ p \ v; \ p \neq []; \text{distinct } p \rrbracket \Longrightarrow \text{cycle } G \ p$

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

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

lemma *countable-cycles* [*simp*]:

assumes *countable* (**V**_{*G*})
shows *countable* (*cycles* *G*)

proof —

have *cycles* *G* $\subseteq$ *lists* **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* $\Rightarrow$ *'v edge* *list*

where *cycle-edges* *p* = *zip* *p* (*rotate1* *p*)

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

by(*auto simp add: cycle-edges-def cycle.simps neq-Nil-conv zip-Cons1 split: list.split*)

lemma *distinct-cycle-edges*:

cycle *G* *p* $\Longrightarrow$ *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*) $\leftarrow$ *cycle-edges* *p*. *x'* = *x*]) = *card* (*set* [(*x'*, *y*) $\leftarrow$ *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'* $\leftarrow$ *map* *fst* (*cycle-edges* *p*). *x'* = *x*]

```

    by(subst distinct-card; simp add: filter-map o-def split-def)
  also have ... = (if  $x \in \text{set } p$  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 ← 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  $(x, y) \in \text{set } (\text{cycle-edges } p)$ 
  shows  $\exists z. (z, x) \in \text{set } (\text{cycle-edges } p)$ 
using assms
by(cases)(auto 4 3 simp add: cycle-edges-def cong: conj-cong split: if-split-asm in-
tro: set-zip-rightI dest: set-zip-leftD)

```

lemma *cycle-edges-edges*:

```

  assumes cycle G p
  shows  $\text{set } (\text{cycle-edges } p) \subseteq \mathbf{E}_G$ 
proof
  fix x
  assume  $x \in \text{set } (\text{cycle-edges } p)$ 
  then obtain i where  $x = (p ! i, \text{rotate1 } p ! i)$  and  $i < \text{length } p$ 
    by(auto simp add: cycle-edges-def set-zip)
  from assms obtain v where  $p: \text{path } G \ v \ p \ v$  and  $p \neq []$  and distinct p by cases
  let ?i = Suc i mod length p
  have  $?i < \text{length } p$  by (simp add:  $\langle p \neq [] \rangle$ )
  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  $\text{Suc } i = \text{length } p$  by simp
    moreover from  $p \langle p \neq [] \rangle$  have  $\text{last } p = v$  by(rule rtrancl-path-last)
    ultimately show ?thesis using  $\langle p \neq [] \rangle$  by(simp add: last-conv-nth)(metis
diff-Suc-Suc diff-zero)
  qed
  also have  $p ! ?i = \text{rotate1 } p ! i$  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 $\Rightarrow$ ennreal
 source :: $'v$
 sink :: $'v$

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

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

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

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

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

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

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

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

lemma *d-OUT-0* [*simp*]: $d\text{-OUT } (\lambda\ -. \ 0)\ x = 0$
by(*simp add: d-OUT-def*)

lemma *d-IN-0* [*simp*]: $d\text{-IN } (\lambda\ -. \ 0)\ x = 0$
by(*simp add: d-IN-def*)

lemma *d-OUT-add*: $d\text{-OUT } (\lambda e. f\ e + g\ e)\ x = d\text{-OUT } f\ x + d\text{-OUT } g\ x$
unfolding *d-OUT-def* **by**(*simp add: nn-integral-add*)

lemma *d-IN-add*: $d\text{-IN } (\lambda e. f\ e + g\ e)\ x = d\text{-IN } f\ x + d\text{-IN } g\ x$
unfolding *d-IN-def* **by**(*simp add: nn-integral-add*)

lemma *d-OUT-cmult*: $d\text{-OUT } (\lambda e. c * f\ e)\ x = c * d\text{-OUT } f\ x$
by(*simp add: d-OUT-def nn-integral-cmult*)

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

lemma *d-OUT-diff*:
assumes $\bigwedge y. g(x, y) \leq f(x, y) \text{ } d\text{-OUT } g x \neq \top$
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 $\bigwedge x. g(x, y) \leq f(x, y) \text{ } d\text{-IN } g y \neq \top$
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*: $(\bigwedge y. (x, y) \notin \mathbf{E} \implies g(x, y) = 0) \implies d\text{-OUT } g x = (\sum^+_{y \in \mathbf{OUT}} x. g(x, y))$
and *d-IN-alt-def*: $(\bigwedge x. (x, y) \notin \mathbf{E} \implies g(x, y) = 0) \implies d\text{-IN } g y = (\sum^+_{x \in \mathbf{IN}} y. g(x, y))$
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*: $d\text{-OUT } g x = (\sum^+_{y \in \{y. (x, y) \in \text{support-flow } g\}}. g(x, y))$
and *d-IN-alt-def2*: $d\text{-IN } g y = (\sum^+_{x \in \{x. (x, y) \in \text{support-flow } g\}}. g(x, y))$
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* :: $('v \text{ edge} \implies \text{ennreal}) \Rightarrow 'v \Rightarrow \text{ennreal}$
where $d\text{-diff } g x = d\text{-OUT } g x - d\text{-IN } g x$

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

inductive-set *SINK* :: ('v edge $\Rightarrow$ ennreal) $\Rightarrow$ 'v set

for *f*

where *SINK*: $d\text{-OUT } f \ x = 0 \implies x \in \text{SINK } f$

lemma *SINK-mono*:

assumes $\bigwedge e. f \ e \leq g \ e$

shows $\text{SINK } g \subseteq \text{SINK } f$

proof(rule subsetI; erule *SINK.cases*; hypsubst)

fix *x*

assume $d\text{-OUT } g \ x = 0$

moreover have $d\text{-OUT } f \ x \leq d\text{-OUT } g \ x$ **using** *assms* **by**(rule *d-OUT-mono*)

ultimately have $d\text{-OUT } f \ x = 0$ **by** *simp*

thus $x \in \text{SINK } f$ **..**

qed

lemma *SINK-mono'*: $f \leq g \implies \text{SINK } g \subseteq \text{SINK } f$

by(rule *SINK-mono*)(rule *le-funD*)

lemma *support-flow-Sup*: $\text{support-flow } (\text{Sup } Y) = (\bigcup_{f \in Y. \text{support-flow } f})$

by(auto simp add: *support-flow-conv less-SUP-iff*)

lemma

assumes *chain*: *Complete-Partial-Order.chain* ($\leq$) *Y*

and *Y*: $Y \neq \{\}$

and *countable*: *countable* ($\text{support-flow } (\text{Sup } Y)$)

shows *d-OUT-Sup*: $d\text{-OUT } (\text{Sup } Y) \ x = (\text{SUP } f \in Y. d\text{-OUT } f \ x)$ (**is** ?*OUT* *x* **is** ?*lhs1* $x = ?\text{rhs1 } x$)

and *d-IN-Sup*: $d\text{-IN } (\text{Sup } Y) \ y = (\text{SUP } f \in Y. d\text{-IN } f \ y)$ (**is** ?*IN* **is** ?*lhs2* $= ?\text{rhs2}$)

and *SINK-Sup*: $\text{SINK } (\text{Sup } Y) = (\bigcap_{f \in Y. \text{SINK } f})$ (**is** ?*SINK*)

proof –

have *chain'*: *Complete-Partial-Order.chain* ($\leq$) $((\lambda f y. f \ (x, y)) \ ' Y)$ **for** *x* **using** *chain*

by(rule *chain-imageI*)(simp add: *le-fun-def*)

have *countable'*: *countable* $\{y. (x, y) \in \text{support-flow } (\text{Sup } Y)\}$ **for** *x*

using - *countable*[*THEN* *countable-image*[**where** $f = \text{snd}$]]

by(rule *countable-subset*)(auto intro: *prod.expand rev-image-eqI*)

{ fix *x*

have ?*lhs1* $x = (\sum^+ y \in \{y. (x, y) \in \text{support-flow } (\text{Sup } Y)\}. \text{SUP } f \in Y. f \ (x, y))$

by(subst *d-OUT-alt-def2*; simp)

also have $\dots = (\text{SUP } f \in Y. \sum^+ y \in \{y. (x, y) \in \text{support-flow } (\text{Sup } Y)\}. f \ (x, y))$ **using** *Y*

by(rule *nn-integral-monotone-convergence-SUP-countable*)(auto simp add: *chain'* intro: *countable'*)

also have $\dots = ?\text{rhs1 } x$ **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 ( $\leq$ ) (( $\lambda f x. f(x, y)$ ) ' Y) for y using
chain
  by(rule chain-imageI)(simp add: le-fun-def)
have countable'': countable {x. (x, y)  $\in$  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 = ( $\sum^+ x \in \{x. (x, y) \in \text{support-flow (Sup Y)}\}. \text{SUP } f \in Y. f(x, y)$ )
  by(subst d-IN-alt-def2; simp)
also have ... = ( $\text{SUP } f \in Y. \sum^+ x \in \{x. (x, y) \in \text{support-flow (Sup Y)}\}. f(x, y)$ ) 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 ( $\leq$ ) Y
  and Y: Y  $\neq$  {}
  and countable: countable (support-flow f)
  and bounded:  $\bigwedge g e. g \in Y \implies g e \leq f e$ 
  shows d-OUT-Inf: d-OUT f x  $\neq$  top  $\implies$  d-OUT (Inf Y) x = (INF g  $\in$  Y. d-OUT
g x) (is -  $\implies$  ?OUT is -  $\implies$  ?lhs1 = ?rhs1)
  and d-IN-Inf: d-IN f x  $\neq$  top  $\implies$  d-IN (Inf Y) x = (INF g  $\in$  Y. d-IN g x) (is -
 $\implies$  ?IN is -  $\implies$  ?lhs2 = ?rhs2)
proof -

We take a detour here via suprema because we have more theorems about
integralN with suprema than with infima.

from Y obtain g0 where g0: g0  $\in$  Y by auto
have g0-le-f: g0 e  $\leq$  f e for e by(rule bounded[OF g0])

have support-flow (SUP g  $\in$  Y. ( $\lambda e. f e - g e$ ))  $\subseteq$  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  $\in$  Y. ( $\lambda e. f e - g e$ )))
by(rule countable-subset)(rule countable)

have Complete-Partial-Order.chain ( $\geq$ ) Y using chain by(simp add: chain-dual)
hence chain': Complete-Partial-Order.chain ( $\leq$ ) (( $\lambda g e. f e - g e$ ) ' Y)
  by(rule chain-imageI)(auto simp add: le-fun-def intro: ennreal-minus-mono)

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

```

```

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

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

have ?lhs2 =  $(\sum^+ y. \text{INF } g \in Y. g \ (y, x))$  by(simp add: d-IN-def)
also have  $\dots = d\text{-IN } f \ x - (\sum^+ y. f \ (y, x) - (\text{INF } g \in Y. g \ (y, x)))$  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  $(\sum^+ y. f \ (y, x) - (\text{INF } g \in Y. g \ (y, x))) = d\text{-IN } (\lambda e. \text{SUP } g \in Y. f$ 
 $e - g \ e) \ x$ 
unfolding d-IN-def by(subst SUP-const-minus-ennreal)(simp-all add: Y)
also have  $\dots = (\text{SUP } h \in (\lambda g \ e. f \ e - g \ e) \ 'Y. d\text{-IN } h \ x)$  using countable'
chain' Y
by(subst d-IN-Sup[symmetric])(simp-all add: SUP-apply[abs-def])
also have  $\dots = (\text{SUP } g \in Y. d\text{-IN } (\lambda e. f \ e - g \ e) \ x)$  unfolding image-image
..
also have  $\dots = (\text{SUP } g \in Y. d\text{-IN } f \ x - d\text{-IN } g \ x)$ 
by(rule SUP-cong[OF refl] d-IN-diff)+(auto intro: bounded simp add: finite2)
also have  $\dots = d\text{-IN } f \ x - ?\text{rhs2}$  by(subst SUP-const-minus-ennreal)(simp-all
add: Y)
also have  $d\text{-IN } f \ x - \dots = ?\text{rhs2}$ 
by(subst diff-diff-ennreal)(simp-all add: finite bounded1)
finally show  $?IN \ . \}$ 
qed

inductive flow :: ('v, 'more) network-scheme  $\Rightarrow$  'v flow  $\Rightarrow$  bool
for  $\Delta$  (structure) and  $f$ 
where
flow:  $\llbracket \bigwedge e. f \ e \leq \text{capacity } \Delta \ e;$ 
 $\bigwedge x. \llbracket x \neq \text{source } \Delta; x \neq \text{sink } \Delta \rrbracket \Longrightarrow \text{KIR } f \ x \rrbracket$ 
 $\Longrightarrow \text{flow } \Delta \ f$ 

lemma flowD-capacity:  $\text{flow } \Delta \ f \Longrightarrow f \ e \leq \text{capacity } \Delta \ e$ 
by(cases e)(simp add: flow.simps)

lemma flowD-KIR:  $\llbracket \text{flow } \Delta \ f; x \neq \text{source } \Delta; x \neq \text{sink } \Delta \rrbracket \Longrightarrow \text{KIR } f \ x$ 
by(simp add: flow.simps)

lemma flowD-capacity-OUT:  $\text{flow } \Delta \ f \Longrightarrow d\text{-OUT } f \ x \leq d\text{-OUT } (\text{capacity } \Delta) \ x$ 
by(rule d-OUT-mono)(erule flowD-capacity)

lemma flowD-capacity-IN:  $\text{flow } \Delta \ f \Longrightarrow d\text{-IN } f \ x \leq d\text{-IN } (\text{capacity } \Delta) \ x$ 
by(rule d-IN-mono)(erule flowD-capacity)

```

abbreviation $value\text{-}flow :: ('v, 'more) network\text{-}scheme \Rightarrow ('v\ edge \Rightarrow ennreal) \Rightarrow ennreal$
where $value\text{-}flow\ \Delta\ f \equiv d\text{-}OUT\ f\ (source\ \Delta)$

5.1 Cut

type-synonym $'v\ cut = 'v\ set$

inductive $cut :: ('v, 'more) network\text{-}scheme \Rightarrow 'v\ cut \Rightarrow bool$
for Δ **and** S
where $cut: \llbracket source\ \Delta \in S; sink\ \Delta \notin S \rrbracket \Longrightarrow cut\ \Delta\ S$

inductive $orthogonal :: ('v, 'more) network\text{-}scheme \Rightarrow 'v\ flow \Rightarrow 'v\ cut \Rightarrow bool$
for $\Delta\ f\ S$
where
 $\llbracket \bigwedge x\ y. \llbracket edge\ \Delta\ x\ y; x \in S; y \notin S \rrbracket \Longrightarrow f\ (x, y) = capacity\ \Delta\ (x, y);$
 $\llbracket \bigwedge x\ y. \llbracket edge\ \Delta\ x\ y; x \notin S; y \in S \rrbracket \Longrightarrow f\ (x, y) = 0 \rrbracket$
 $\Longrightarrow orthogonal\ \Delta\ f\ S$

lemma $orthogonalD\text{-}out:$
 $\llbracket orthogonal\ \Delta\ f\ S; edge\ \Delta\ x\ y; x \in S; y \notin S \rrbracket \Longrightarrow f\ (x, y) = capacity\ \Delta\ (x, y)$
by($simp\ add: orthogonal.simps$)

lemma $orthogonalD\text{-}in:$
 $\llbracket orthogonal\ \Delta\ f\ S; edge\ \Delta\ x\ y; x \notin S; y \in S \rrbracket \Longrightarrow f\ (x, y) = 0$
by($simp\ add: orthogonal.simps$)

5.2 Countable network

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

lemma $sink\text{-}neq\text{-}source\ [simp]: sink\ \Delta \neq source\ \Delta$
using $source\text{-}neq\text{-}sink[symmetric]$.

lemma $countable\text{-}V\ [simp]: countable\ \mathbf{V}$
unfolding $\mathbf{V}\text{-}def$ **using** $countable\text{-}E$ **by** $auto$

lemma $flowD\text{-}outside:$
assumes $g: flow\ \Delta\ g$
shows $e \notin \mathbf{E} \Longrightarrow g\ e = 0$
using $flowD\text{-}capacity[OF\ g, of\ e]\ capacity\text{-}outside[of\ e]$ **by** $simp$

lemma $flowD\text{-}finite:$

```

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

lemma zero-flow [simp]: flow  $\Delta$  ( $\lambda \cdot 0$ )
by(rule flow.intros) simp-all

end

```

5.3 Reduction for avoiding antiparallel edges

```

locale antiparallel-edges = countable-network  $\Delta$ 
  for  $\Delta :: ('v, 'more)$  network-scheme (structure)
begin

```

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

```

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

```

```

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

```

```

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

```

```

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

```

```

lemma split-Vertex1-eq-0I: ( $\bigwedge z. y \neq \text{Edge } x z$ )  $\Longrightarrow$  split f (Vertex x, y) = 0
by(cases y) auto

```

```

lemma split-Vertex2-eq-0I: ( $\bigwedge z. y \neq \text{Edge } z x$ )  $\Longrightarrow$  split f (y, Vertex x) = 0
by(cases y) simp-all

```

```

lemma split-Edge1-eq-0I: ( $\bigwedge z. y \neq \text{Vertex } x$ )  $\Longrightarrow$  split f (Edge z x, y) = 0
by(cases y) simp-all

```

```

lemma split-Edge2-eq-0I: ( $\bigwedge z. y \neq \text{Vertex } x$ )  $\Longrightarrow$  split f (y, Edge x z) = 0
by(cases y) simp-all

```


definition $\Delta'' :: 'v$ vertex network

where $\Delta'' = \langle \text{edge} = \text{edg}, \text{capacity} = \text{split} (\text{capacity } \Delta), \text{source} = \text{Vertex} (\text{source } \Delta), \text{sink} = \text{Vertex} (\text{sink } \Delta) \rangle$

lemma $\Delta''\text{-sel}$ [simp]:

$\text{edge } \Delta'' = \text{edg}$
 $\text{capacity } \Delta'' = \text{split} (\text{capacity } \Delta)$
 $\text{source } \Delta'' = \text{Vertex} (\text{source } \Delta)$
 $\text{sink } \Delta'' = \text{Vertex} (\text{sink } \Delta)$

by(simp-all add: $\Delta''\text{-def}$)

lemma $\mathbf{E}\text{-}\Delta''$: $\mathbf{E}_{\Delta''} = (\lambda(x, y). (\text{Vertex } x, \text{Edge } x y)) \text{ ' } \mathbf{E} \cup (\lambda(x, y). (\text{Edge } x y, \text{Vertex } y)) \text{ ' } \mathbf{E}$

by(auto elim: edg.cases)

lemma $\mathbf{V}\text{-}\Delta''$: $\mathbf{V}_{\Delta''} = \text{Vertex} \text{ ' } \mathbf{V} \cup \text{case-prod } \text{Edge} \text{ ' } \mathbf{E}$

by(auto 4 4 simp add: vertex-def elim!: edg.cases)

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

by(simp add: inj-on-def)

lemma inj-on-Edge2 [simp]: inj-on $(\text{Edge } x)$ A

by(simp add: inj-on-def)

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

proof(rule trans)

show ?lhs = $(\sum^+ v' \in \text{range } (\lambda y. \text{Edge } y x). \text{split } f (v', \text{Vertex } x))$

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 $(\text{split } f)$ $(\text{Vertex } x) = \text{d-OUT } f x$ (is ?lhs = ?rhs)

proof(rule trans)

show ?lhs = $(\sum^+ v' \in \text{range } (\text{Edge } x). \text{split } f (\text{Vertex } x, v'))$

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 $(\text{split } f)$ $(\text{Edge } x y) = \text{max } 0 (f (x, y))$ (is ?lhs = ?rhs)

proof(rule trans)

show ?lhs = $(\sum^+ v'. \text{split } f (v', \text{Edge } x y) * \text{indicator } \{\text{Vertex } x\} v')$

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]: $d\text{-OUT } (\text{split } f) (\text{Edge } x \ y) = \max 0 \ (f \ (x, y))$
(is ?lhs = ?rhs)
proof(rule trans)
 show ?lhs = $(\sum^+ v'. \text{split } f \ (\text{Edge } x \ y, v') * \text{indicator } \{\text{Vertex } y\} \ v')$
 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** $\mathbf{E}\text{-}\Delta''$ **by**(simp)
 show source $\Delta'' \neq$ sink Δ'' **by** auto
 show capacity $\Delta'' \ e = 0$ 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 $\Delta'' \ e \neq$ 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 $\Delta \ f$
 shows flow $\Delta'' \ (\text{split } f)$
proof
 show split $f \ e \leq$ capacity $\Delta'' \ e$ **for** e
 by(cases (f , e) rule: split.cases)(auto intro: flowD-capacity[OF assms] intro:
SUP-upper2 assms)
 show KIR ($\text{split } f$) x if $x \neq$ source $\Delta'' \ x \neq$ sink Δ'' **for** x
 using that **by**(cases x)(auto dest: flowD-KIR[OF assms])
qed

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

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

```

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

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

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

lemma value-collect:  $\text{flow } \Delta'' \ f \implies \text{value-flow } \Delta \ (\text{collect } f) = \text{value-flow } \Delta'' \ f$ 
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* $\Rightarrow$ *ennreal*

inductive *current* :: (*'v*, *'more*) *web-scheme* $\Rightarrow$ *'v current* $\Rightarrow$ *bool*
for Γ f
where

current:
 $\llbracket \bigwedge x. d\text{-OUT } f\ x \leq \text{weight } \Gamma\ x;$
 $\bigwedge x. d\text{-IN } f\ x \leq \text{weight } \Gamma\ x;$
 $\bigwedge x. x \notin A\ \Gamma \Longrightarrow d\text{-OUT } f\ x \leq d\text{-IN } f\ x;$
 $\bigwedge a. a \in A\ \Gamma \Longrightarrow d\text{-IN } f\ a = 0;$
 $\bigwedge b. b \in B\ \Gamma \Longrightarrow d\text{-OUT } f\ b = 0;$
 $\bigwedge e. e \notin \mathbf{E}_\Gamma \Longrightarrow f\ e = 0 \rrbracket$
 $\Longrightarrow \text{current } \Gamma\ f$

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

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

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

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

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

lemma *currentD-outside*: $\llbracket \text{current } \Gamma\ f; \neg \text{edge } \Gamma\ x\ y \rrbracket \Longrightarrow f\ (x, y) = 0$
by(*blast elim: current.cases*)

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

lemma *currentD-OUT-eq-0*:
assumes *current* $\Gamma\ f$
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* $\Gamma\ f$
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  $\Gamma$  (structure)
  assumes current  $\Gamma$  f
  shows support-flow  $f \subseteq \mathbf{E}$ 
using currentD-outside[OF assms] by(auto simp add: support-flow.simps intro: ccontr)

lemma currentD-outside-IN:  $\llbracket \text{current } \Gamma f; x \notin \mathbf{V}_\Gamma \rrbracket \implies d\text{-IN } f x = 0$ 
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:  $\llbracket \text{current } \Gamma f; x \notin \mathbf{V}_\Gamma \rrbracket \implies d\text{-OUT } f x = 0$ 
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  $\Gamma$  h  $\implies h(x, y) \leq \text{weight } \Gamma y$ 
by (metis order-trans d-IN-ge-point currentD-weight-IN)

lemma currentD-weight-out: current  $\Gamma$  h  $\implies h(x, y) \leq \text{weight } \Gamma x$ 
by (metis order-trans d-OUT-ge-point currentD-weight-OUT)

lemma current-leI:
  fixes  $\Gamma$  (structure)
  assumes f: current  $\Gamma$  f
  and le:  $\bigwedge e. g e \leq f e$ 
  and OUT-IN:  $\bigwedge x. x \notin A \Gamma \implies d\text{-OUT } g x \leq d\text{-IN } g x$ 
  shows current  $\Gamma$  g
proof
  show d-OUT  $g x \leq \text{weight } \Gamma x$  for x
    using d-OUT-mono[of g x f, OF le] currentD-weight-OUT[OF f] by(rule order-trans)
  show d-IN  $g x \leq \text{weight } \Gamma x$  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  $e \notin \mathbf{E}$  for e
    using currentD-outside'[OF f that] le[of e] by simp
qed(blast intro: OUT-IN)+

lemma current-weight-mono:
   $\llbracket \text{current } \Gamma f; \text{edge } \Gamma = \text{edge } \Gamma'; A \Gamma = A \Gamma'; B \Gamma = B \Gamma'; \bigwedge x. \text{weight } \Gamma x \leq \text{weight } \Gamma' x \rrbracket$ 
   $\implies \text{current } \Gamma' f$ 
by(auto 4 3 elim!: current.cases intro!: current.intros intro: order-trans)

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

```

```

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

lemma current-0 [simp]: current  $\Gamma$  zero-current
by(auto simp add: current.simps)

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

lemma web-flowD-current: web-flow  $\Gamma f \Longrightarrow \text{current } \Gamma f$ 
by(erule web-flow.cases)

lemma web-flowD-KIR:  $\llbracket \text{web-flow } \Gamma f; x \notin A \Gamma; x \notin B \Gamma \rrbracket \Longrightarrow \text{KIR } f x$ 
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  $\Gamma$  f
where
  A:  $x \in A \Gamma \Longrightarrow x \in \text{SAT } \Gamma f$ 
  | IN:  $d\text{-IN } f x \geq \text{weight } \Gamma x \Longrightarrow x \in \text{SAT } \Gamma f$ 
  — We use  $\geq \text{weight}$  such that SAT is monotone w.r.t. increasing currents

```

```

lemma SAT-0 [simp]: SAT  $\Gamma$  zero-current =  $A \Gamma \cup \{x. \text{weight } \Gamma x \leq 0\}$ 
by(auto simp add: SAT.simps)

```

```

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

```

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

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

abbreviation *terminal* :: $('v, 'more) \text{ web-scheme} \Rightarrow 'v \text{ current} \Rightarrow 'v \text{ set} (\langle \text{TER}_1 \rangle)$
where *terminal* $\Gamma f \equiv \text{SAT } \Gamma f \cap \text{SINK } f$

6.2 Separation

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

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

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

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

lemma *separatingD*:
 $\bigwedge A B. \llbracket \text{separating-gen } G A B S; \text{path } G x p y; x \in A; y \in B \rrbracket \implies (\exists z \in \text{set } p. z \in S) \vee x \in S$
by(*blast elim: separating-gen.cases*)

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

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

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

abbreviation *essential-web* :: $('v, 'more) \text{ web-scheme} \Rightarrow 'v \text{ set} \Rightarrow 'v \text{ set} (\langle \mathcal{E}_1 \rangle)$

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

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

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

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

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

lemma *essential-vertex*: $\bigwedge B. \llbracket \text{essential } G \ B \ S \ x; x \notin B \rrbracket \implies \text{vertex } G \ x$
by(*auto elim!: essentialE simp add: vertex-def elim: rtrancl-path.cases*)

lemma *essential-BI*: $\bigwedge B. x \in B \implies \text{essential } G \ B \ S \ x$
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** $\text{path } \Gamma \ x \ p \ y \ y \in B \ \Gamma \ \bigwedge z. \llbracket x \neq y; z \in \text{set } p \rrbracket \implies z = x \vee z \notin S$
using *assms* **by**(*auto elim: essentialE*)

lemma *essential-mono*: $\bigwedge B. \llbracket \text{essential } G \ B \ S \ x; S' \subseteq S \rrbracket \implies \text{essential } G \ B \ S' \ x$
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* $G \ A \ B \ \{x \in S. \text{essential } G \ B \ S \ x\}$ (**is** *separating-gen* - - - ?E)
proof
fix $x \ y \ p$
assume $x: x \in A$ **and** $y: y \in B$ **and** $p: \text{path } G \ x \ p \ y$
from *separatingD*[*OF assms* $p \ x \ y$] **have** $\exists z \in \text{set } (x \ \# \ p). z \in S$ **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*: $\bigwedge z. z \in \text{set } zs \implies z \notin S$ **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  $(\exists z \in set \ p. \ z \in \ ?E) \vee x \in \ ?E$ 
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:  $\bigwedge B. \ roofed-gen \ G \ B \ S = \{x. \ \forall p. \ \forall y \in B. \ path \ G \ x \ p \ y \longrightarrow$ 
 $(\exists z \in set \ p. \ z \in S) \vee x \in S\}$ 

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

abbreviation roofed-network :: ('v, 'more) network-scheme  $\Rightarrow$  'v set  $\Rightarrow$  'v set
 $(\langle RF^N 1 \rangle)$ 
where roofed-network  $\Delta \equiv roofed-gen \ \Delta \ \{sink \ \Delta\}$ 

lemma roofedI [intro?]:
 $\bigwedge B. (\bigwedge p \ y. \ll path \ G \ x \ p \ y; \ y \in B \rr \Longrightarrow (\exists z \in set \ p. \ z \in S) \vee x \in S) \Longrightarrow x \in$ 
 $roofed-gen \ G \ B \ S$ 
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:  $\bigwedge B. \ll x \in roofed-gen \ G \ B \ S; \ S \subseteq S' \rr \Longrightarrow x \in roofed-gen$ 
 $G \ B \ S'$ 
using roofed-mono[THEN subsetD] .

```

lemma *roofedD*: $\bigwedge B. \llbracket x \in \text{roofed-gen } G \ B \ S; \text{path } G \ x \ p \ y; y \in B \rrbracket \implies (\exists z \in \text{set } p. z \in S) \vee x \in S$

unfolding *roofed-def* **by** *blast*

lemma *separating-RF-A*:

fixes *A B*

assumes *separating-gen G A B X*

shows $A \subseteq \text{roofed-gen } G \ B \ X$

by(*rule subsetI* *roofedI*)+(*erule separatingD*[*OF assms*])

lemma *roofed-idem*: **fixes** *B* **shows** $\text{roofed-gen } G \ B \ (\text{roofed-gen } G \ B \ S) = \text{roofed-gen } G \ B \ S$

proof(*rule equalityI subsetI* *roofedI*)+

fix *x p y*

assume *x: x ∈ roofed-gen G B (roofed-gen G B S) and p: path G x p y and y: y ∈ B*

from *roofedD*[*OF x p y*] **obtain** *z* **where** *z: z ∈ set (x # p) and z: z ∈ 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** $(\exists z \in \text{set } p. z \in S) \vee x \in S$

by(*auto simp add: Cons-eq-append-conv*)

qed(*rule roofed-mono roofed-greater*)+

lemma *in-roofed-mono'*: $\bigwedge B. \llbracket x \in \text{roofed-gen } G \ B \ S; S \subseteq \text{roofed-gen } G \ B \ S' \rrbracket \implies x \in \text{roofed-gen } G \ B \ S'$

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

lemma *roofed-mono'*: $\bigwedge B. S \subseteq \text{roofed-gen } G \ B \ S' \implies \text{roofed-gen } G \ B \ S \subseteq \text{roofed-gen } G \ B \ S'$

by(*rule subsetI*)(*rule in-roofed-mono'*)

lemma *roofed-idem-Un1*: **fixes** *B* **shows** $\text{roofed-gen } G \ B \ (\text{roofed-gen } G \ B \ S \cup T) = \text{roofed-gen } G \ B \ (S \cup T)$

proof –

have $S \subseteq T \cup \text{roofed-gen } G \ B \ S$

by (*metis* (*no-types*) *UnCI* *roofed-greater subsetCE subsetI*)

then have $S \cup T \subseteq T \cup \text{roofed-gen } G \ B \ S \wedge T \cup \text{roofed-gen } G \ B \ S \subseteq \text{roofed-gen } G \ B \ (S \cup T)$

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

```

proof(rule equalityI)
  show ?rhs  $\subseteq$  ?lhs by(rule roofed-mono)(blast intro: roofed-greaterI)
  show ?lhs  $\subseteq$  ?rhs by(rule roofed-mono')(blast intro: in-roofed-mono)
qed

lemma RF-essential: fixes  $\Gamma$  (structure) shows  $RF (\mathcal{E} S) = RF S$ 
proof(intro set-eqI iffI)
  fix x
  assume  $RF: x \in RF S$ 
  show  $x \in RF (\mathcal{E} S)$ 
  proof
    fix p y
    assume p: path  $\Gamma$  x p y and y:  $y \in B \Gamma$ 
    from roofedD[OF RF this] 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:  $\bigwedge z. z \in set zs \implies z \notin S$  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  $(\exists z \in set p. z \in \mathcal{E} S) \vee x \in \mathcal{E} S$ 
    proof(cases)
      case empty
      hence  $x \in \mathcal{E} S$  using z p last y by(auto simp add: essential-def)
      thus ?thesis ..
    next
      case (Cons  $ys'$ )
      from p have path  $\Gamma$  z zs y unfolding Cons by(rule rtrancl-path-appendE)
      hence  $z \in \mathcal{E} S$  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  $\Gamma$  (structure) and B
  assumes essential  $\Gamma B S x$ 
  obtains p y where path  $\Gamma$  x p y y  $\in B$  distinct  $(x \# p) \bigwedge z. z \in set p \implies z \notin$ 
  roofed-gen  $\Gamma B S$ 
proof –
  from assms obtain p y where p: path  $\Gamma$  x p y and y:  $y \in B$ 
  and bypass:  $\bigwedge z. [x \neq y; z \in set p] \implies z = x \vee z \notin S$  by(rule essentialE)
  blast
  from p obtain p' where p': path  $\Gamma$  x p' y and distinct: distinct  $(x \# p')$ 
  and subset:  $set p' \subseteq set p$  by(rule rtrancl-path-distinct)
  { fix z
    assume z:  $z \in set p'$ 
    hence  $y \in set p'$  using rtrancl-path-last[OF p', symmetric] p'
    by(auto elim: rtrancl-path.cases intro: last-in-set)
  }

```

```

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

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

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

lemma in-roofed-essentialD:
  fixes  $\Gamma$  (structure)
  assumes  $RF: x \in RF \ S$ 
  and ess: essential  $\Gamma \ (B \ \Gamma) \ S \ x$ 
  shows  $x \in S$ 
proof –
  from ess obtain  $p \ y$  where  $p$ : path  $\Gamma \ x \ p \ y$  and  $y: y \in B \ \Gamma$  and distinct  $(x \# p)$ 
  and bypass:  $\wedge z. z \in \text{set } p \implies z \notin S$  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  $\Gamma$  (structure) shows separating  $\Gamma \ (RF \ S) \longleftrightarrow separating \ \Gamma \ S$ 
proof
  assume sep: separating  $\Gamma \ (RF \ S)$ 
  show separating  $\Gamma \ S$ 
  proof
    fix  $x \ y \ p$ 
    assume  $p$ : path  $\Gamma \ x \ p \ y$  and  $x: x \in A \ \Gamma$  and  $y: y \in B \ \Gamma$ 
    from separatingD[OF sep p x y] have  $\exists z \in \text{set } (x \# p). z \in RF \ S$  by auto
    from split-list-last-prop[OF this] obtain ys z zs where split:  $x \# p = \text{ys} @ z \# \text{zs}$ 
    and  $z: z \in RF \ S$  and bypass:  $\wedge z'. z' \in \text{set } \text{zs} \implies z' \notin RF \ S$  by auto
    from p split have path  $\Gamma \ z \ \text{zs} \ y$  by (cases ys) (auto elim: rtrancl-path-appendE)
    hence essential  $\Gamma \ (B \ \Gamma) \ S \ z$  using y
  
```

```

    by(rule essentialI)(auto dest: bypass intro: roofed-greaterI)
  with z have z ∈ S by(rule in-roofed-essentialD)
  with split show (∃ z ∈ set p. z ∈ S) ∨ x ∈ S by(cases ys)auto
qed
qed(blast intro: roofed-greaterI separating-weakening)

```

definition *roofed-circ* :: ('v, 'more) web-scheme ⇒ 'v set ⇒ 'v set (↪RF^o↩)

where *roofed-circ* Γ S = *roofed* Γ S - $\mathcal{E}_\Gamma$ S

lemma *roofed-circI*: **fixes** Γ (**structure**) **shows**

$\llbracket x \in RF\ T; x \in T \implies \neg \text{essential}\ \Gamma\ (B\ \Gamma)\ T\ x \rrbracket \implies x \in RF^\circ\ T$

by(*simp add: roofed-circ-def*)

lemma *roofed-circE*:

fixes Γ (**structure**)

assumes $x \in RF^\circ\ T$

obtains $x \in RF\ T \neg \text{essential}\ \Gamma\ (B\ \Gamma)\ T\ x$

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* Γ (B Γ) ($\mathcal{E}\ S$) = *essential* Γ (B Γ) S

by(*subst essential-RF[symmetric]*)(*simp only: RF-essential essential-RF*)

lemma *RF-in-B*: **fixes** Γ (**structure**) **shows** $x \in B\ \Gamma \implies x \in RF\ S \longleftrightarrow x \in S$

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-scheme* $\Rightarrow 'v\ current \Rightarrow bool$
for Γ (**structure**) **and** f

where

wave:
 $\llbracket separating\ \Gamma\ (TER\ f);$
 $\bigwedge x. x \notin RF\ (TER\ f) \implies d-OUT\ f\ x = 0 \rrbracket$
 $\implies wave\ \Gamma\ f$

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

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

lemma *waveD-OUT*: $\llbracket wave\ \Gamma\ f; x \notin RF_{\Gamma}\ (TER_{\Gamma}\ f) \rrbracket \implies d-OUT\ f\ x = 0$
by(*simp add: wave.simps*)

lemma *wave-A-in-RF*: **fixes** Γ (**structure**)
shows $\llbracket wave\ \Gamma\ f; x \in A\ \Gamma \rrbracket \implies x \in RF\ (TER\ f)$
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 $d-IN\ f\ x = 0$

proof —

from x **obtain** $p\ z$ **where** $z: z \in B\ \Gamma$ **and** $p: path\ \Gamma\ x\ p\ z$
and *bypass*: $\bigwedge z. z \in set\ p \implies z \notin TER\ f\ x \notin TER\ f$

```

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

lemma current-Sup:
  fixes  $\Gamma$  (structure)
  assumes chain: Complete-Partial-Order.chain  $(\leq) Y$ 
  and  $Y: Y \neq \{\}$ 
  and current:  $\bigwedge f. f \in Y \implies \text{current } \Gamma f$ 
  and countable [simp]: countable (support-flow (Sup Y))
  shows  $\text{current } \Gamma (\text{Sup } Y)$ 
proof(rule, goal-cases)
  case (1 x)
  have  $d\text{-OUT } (\text{Sup } Y) x = (\text{SUP } f \in Y. d\text{-OUT } f x)$  using chain Y by(simp add: d-OUT-Sup)
  also have  $\dots \leq \text{weight } \Gamma x$  using 1
  by(intro SUP-least)(auto dest!: current currentD-weight-OUT)
  finally show ?case .
next
  case (2 x)
  have  $d\text{-IN } (\text{Sup } Y) x = (\text{SUP } f \in Y. d\text{-IN } f x)$  using chain Y by(simp add: d-IN-Sup)
  also have  $\dots \leq \text{weight } \Gamma x$  using 2
  by(intro SUP-least)(auto dest!: current currentD-weight-IN)
  finally show ?case .
next
  case (3 x)
  have  $d\text{-OUT } (\text{Sup } Y) x = (\text{SUP } f \in Y. d\text{-OUT } f x)$  using chain Y by(simp add: d-OUT-Sup)
  also have  $\dots \leq (\text{SUP } f \in Y. d\text{-IN } f x)$  using 3
  by(intro SUP-mono)(auto dest: current currentD-OUT-IN)
  also have  $\dots = d\text{-IN } (\text{Sup } Y) x$  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 ∉ E
  from currentD-outside'[OF current this] have f e = 0 if f ∈ 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 ≠ {}
  and wave: ⋀f. f ∈ Y ⇒ 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 ∈ B Γ
    define P where P = {x} ∪ set p

    let ?f = λf. 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 ∈ 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 ∈ Y f ≤ 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  $\langle g \in Y \rangle$ , of  $\Gamma$ ] SINK-Sup[OF chain  $Y$  countable]
  by blast }
  with  $f$  have  $\exists f \in Y. \forall g \in Y. g \geq f \longrightarrow \text{TER } g \cap P \subseteq \text{TER } (\text{Sup } Y) \cap P$  by
blast }
  note subset = this

show separating  $\Gamma$  ( $\text{TER } (\text{Sup } Y)$ )
proof
  fix  $x \ y \ p$ 
  assume *: path  $\Gamma \ x \ p \ y \ y \in B \ \Gamma$  and  $x \in A \ \Gamma$ 
  let  $?P = \{x\} \cup \text{set } p$ 
  from subset[OF *] obtain  $f$  where  $f: f \in Y$ 
    and subset:  $\text{TER } f \cap ?P \subseteq \text{TER } (\text{Sup } Y) \cap ?P$  by blast
  from wave[OF f] have  $\text{TER } f \cap ?P \neq \{\}$  using *  $\langle x \in A \ \Gamma \rangle$ 
    by(auto simp add: wave.simps dest: separatingD)
  with subset show  $(\exists z \in \text{set } p. z \in \text{TER } (\text{Sup } Y)) \vee x \in \text{TER } (\text{Sup } Y)$  by blast
qed

fix  $x$ 
assume  $x \notin \text{RF } (\text{TER } (\text{Sup } Y))$ 
then obtain  $p \ y$  where  $y: y \in B \ \Gamma$ 
  and  $p$ : path  $\Gamma \ x \ p \ y$ 
  and  $\text{ter}$ :  $\text{TER } (\text{Sup } Y) \cap (\{x\} \cup \text{set } p) = \{\}$  by(auto simp add: roofed-def)
let  $?P = \{x\} \cup \text{set } p$ 
from subset[OF  $p \ y$ ] obtain  $f$  where  $f: f \in Y$ 
  and subset:  $\bigwedge g. [\![ g \in Y; f \leq g ]\!] \implies \text{TER } g \cap ?P \subseteq \text{TER } (\text{Sup } Y) \cap ?P$  by
blast

{ fix  $g$ 
  assume  $g: g \in Y$ 
  with chain  $f$  have  $f \leq g \vee g \leq f$  by(rule chainD)
  hence d-OUT  $g \ x = 0$ 
  proof
    assume  $f \leq g$ 
    from subset[OF  $g$  this]  $\text{ter}$  have  $\text{TER } g \cap ?P = \{\}$  by blast
    with  $p \ y$  have  $x \notin \text{RF } (\text{TER } g)$  by(auto simp add: roofed-def)
    with wave[OF  $g$ ] show ?thesis by(blast elim: wave.cases)
  next
    assume  $g \leq f$ 
    from subset  $\text{ter } f$  have  $\text{TER } f \cap ?P = \{\}$  by blast
    with  $y \ p$  have  $x \notin \text{RF } (\text{TER } f)$  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  $g \ x \leq \text{d-OUT } f \ x$  using  $\langle g \leq f \rangle$  [THEN le-funD]
  by(rule d-OUT-mono)
  ultimately show ?thesis by simp
qed }
thus d-OUT ( $\text{Sup } Y$ )  $x = 0$  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 $\exists f. \text{current } \Gamma f \wedge \text{wave } \Gamma f \wedge (\forall w. \text{current } \Gamma w \wedge \text{wave } \Gamma w \wedge f \leq w \longrightarrow f = w)$
proof —
define *Field-r* **where** *Field-r* = $\{f. \text{current } \Gamma f \wedge \text{wave } \Gamma f\}$
define *r* **where** $r = \{(f, g). f \in \text{Field-r} \wedge g \in \text{Field-r} \wedge f \leq g\}$
have *Field-r*: *Field* $r = \text{Field-r}$ **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 $\exists m \in \text{Field } r. \forall a \in \text{Field } r. (m, a) \in r \longrightarrow a = m$
proof(*rule Zorns-po-lemma*)
fix *Y*
assume $Y \in \text{Chains } r$
hence *Y*: *Complete-Partial-Order.chain* ($\leq$) *Y*
and $w: \bigwedge f. f \in Y \implies \text{wave } \Gamma f$
and $f: \bigwedge f. f \in Y \implies \text{current } \Gamma f$
by(*auto simp add: Chains-def r-def chain-def Field-r-def*)
show $\exists w \in \text{Field } r. \forall f \in Y. (f, w) \in r$
proof(*cases* $Y = \{\}$)
case *True*
have $\text{zero-current} \in \text{Field } r$ **by**(*simp add: Field-r Field-r-def*)
with *True* **show** ?thesis **by** blast
next
case *False*
have *support-flow* (*Sup* *Y*) $\subseteq \mathbf{E}$ **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 countable-subset*)
with $Y \text{ False } f w$ **have** *Sup* *Y* $\in \text{Field } r$ **unfolding** *Field-r* *Field-r-def*
by(*blast intro: wave-lub current-Sup*)
moreover then **have** $(f, \text{Sup } Y) \in r$ **if** $f \in Y$ **for** *f* **using** $w[OF \text{ that}] f[OF \text{ that}]$ **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* Γg **and** *w*: *wave* Γg
and *le*: $\bigwedge e. f e \leq g e$
and *x*: $x \in \mathcal{E} (\text{TER } g)$
shows *essential* $\Gamma (B \Gamma) (\text{TER } f) x$
proof —

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

lemma *essential-eq-leI*:
fixes Γ (**structure**)
assumes g : *current* $\Gamma\ g$ **and** w : *wave* $\Gamma\ g$
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* $\Gamma\ (B\ \Gamma)\ (TER\ f)\ x$ **using** $g\ w\ le\ x$ **by**(*rule* *essen-*
tial-leI)
ultimately show $x \in \mathcal{E}\ (TER\ f)$ **by** *simp*
qed

show $\dots \subseteq \mathcal{E}\ (TER\ g)$
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*: $d\text{-OUT } g\ x = 0$ **and** *IN*: $d\text{-IN } g\ x = 0$ **by**(*rule* *waveD-OUT*
wave-not-RF-IN-zero[*OF g*])+
with x **have** $x \notin A\ \Gamma\ \text{weight } \Gamma\ x > 0$ **by**(*auto* *intro!*: *roofed-greaterI simp*
add: *SAT.simps SINK.simps*)
moreover from *IN* $d\text{-IN-mono}$ [*of f x g, OF le*] **have** $d\text{-IN } f\ x \leq 0$ **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 $RF\text{-essential}$ by(*simp-all add: roofed-circ-def*)
 from x obtain $p\ y$ where $p: path\ \Gamma\ x\ p\ y$ and $y: y \in B\ \Gamma$ and *distinct:*
distinct ($x \# p$)
 and *bypass:* $\bigwedge z. z \in set\ p \implies z \notin RF (TER\ f)$ 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:

$\llbracket a \in A\ \Gamma; a \notin \mathcal{E} (TER\ f); d\text{-OUT}\ f\ a < weight\ \Gamma\ a; \varepsilon < weight\ \Gamma\ a - d\text{-OUT}\ f\ a \rrbracket \implies hindrance\text{-by}\ \Gamma\ f\ \varepsilon$

inductive *hindrance* :: ($'v, 'more$) *web-scheme* $\Rightarrow 'v\ current \Rightarrow bool$
 for Γ (**structure**) and f

where

hindrance:

$\llbracket a \in A\ \Gamma; a \notin \mathcal{E} (TER\ f); d\text{-OUT}\ f\ a < weight\ \Gamma\ a \rrbracket \implies hindrance\ \Gamma\ f$

inductive *hindered* :: ($'v, 'more$) *web-scheme* $\Rightarrow bool$
 for Γ (**structure**)

where *hindered*: $\llbracket hindrance\ \Gamma\ f; current\ \Gamma\ f; wave\ \Gamma\ f \rrbracket \implies hindered\ \Gamma$

inductive *hindered-by* :: ($'v, 'more$) *web-scheme* $\Rightarrow ennreal \Rightarrow bool$
 for Γ (**structure**) and ε

where *hindered-by*: $\llbracket hindrance\text{-by}\ \Gamma\ f\ \varepsilon; current\ \Gamma\ f; wave\ \Gamma\ f \rrbracket \implies hindered\text{-by}\ \Gamma\ \varepsilon$

lemma *hindrance-into-hindrance-by*:

assumes *hindrance* $\Gamma\ f$

shows $\exists \varepsilon > 0. hindrance\text{-by}\ \Gamma\ f\ \varepsilon$

using *assms*

proof *cases*

case (*hindrance* a)

let $? \varepsilon = if\ weight\ \Gamma\ a = \top\ then\ 1\ else\ (weight\ \Gamma\ a - d\text{-OUT}\ f\ a) / 2$

from $\langle d\text{-OUT}\ f\ a < weight\ \Gamma\ a \rangle$ **have** $weight\ \Gamma\ a - d\text{-OUT}\ f\ a > 0$ *weight* $\Gamma\ a$

```

≠ ⊤ ⇒ weight Γ a - d-OUT f a < ⊤
  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 ε ⇒ hindrance Γ f
by(blast elim: hindrance-by.cases intro: hindrance.intros)

lemma hindrance-conv-hindrance-by: hindrance Γ f ⇔ (∃ ε > 0. hindrance-by Γ f
ε)
by(blast intro: hindrance-into-hindrance-by hindrance-by-into-hindrance)

lemma hindered-into-hindered-by: hindered Γ ⇒ ∃ ε > 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 ε ⇒ hindered Γ
by(blast elim: hindered-by.cases intro: hindered.intros dest: hindrance-by-into-hindrance)

lemma hindered-conv-hindered-by: hindered Γ ⇔ (∃ ε > 0. hindered-by Γ ε)
by(blast intro: hindered-into-hindered-by hindered-by-into-hindered)

inductive loose :: ('v, 'more) web-scheme ⇒ bool
  for Γ
where
  loose: [⋀ f. [current Γ f; wave Γ f] ⇒ f = zero-current; ¬ hindrance Γ
zero-current]
    ⇒ loose Γ

lemma looseD-hindrance: loose Γ ⇒ ¬ hindrance Γ zero-current
by(simp add: loose.simps)

lemma looseD-wave:
  [loose Γ; current Γ f; wave Γ f] ⇒ f = zero-current
by(simp add: loose.simps)

lemma loose-unhindered:
  fixes Γ (structure)
  assumes loose Γ
  shows ¬ 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  $\Gamma \Gamma' :: ('v, 'more) \text{ web-scheme}$ 
  assumes [simp]:  $\text{edge } \Gamma = \text{edge } \Gamma' \ A \ \Gamma = A \ \Gamma' \ B \ \Gamma = B \ \Gamma'$ 
  and  $\text{weight-eq}: \bigwedge x. x \notin A \ \Gamma' \implies \text{weight } \Gamma \ x = \text{weight } \Gamma' \ x$ 
  and  $\text{weight-le}: \bigwedge a. a \in A \ \Gamma' \implies \text{weight } \Gamma \ a \geq \text{weight } \Gamma' \ a$ 
begin

private lemma  $\text{essential-eq}: \text{essential } \Gamma = \text{essential } \Gamma'$ 
by(simp add: fun-eq-iff essential-def)

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

qualified lemma  $\text{separating-eq}: \text{separating-gen } \Gamma = \text{separating-gen } \Gamma'$ 
by(intro ext iffI; rule separating-gen.intros; drule separatingD; simp)

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

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

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

lemma  $\text{hindered-mono-web}: \text{hindered } \Gamma' \implies \text{hindered } \Gamma$ 
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*:

$\llbracket \bigwedge x. \llbracket x \in S; x \notin A \Gamma \rrbracket \implies weight \Gamma x \leq d-IN f x;$

$\bigwedge x. \llbracket x \in S; x \in A \Gamma; x \notin B \Gamma \rrbracket \implies d-OUT f x = weight \Gamma x;$

$\bigwedge u v. \llbracket v \in RF S; u \notin RF^\circ S \rrbracket \implies f(u, v) = 0 \rrbracket$

$\implies orthogonal-current \Gamma f S$

lemma *orthogonal-currentD-SAT*: $\llbracket orthogonal-current \Gamma f S; x \in S \rrbracket \implies x \in SAT \Gamma f$

by(auto elim!: *orthogonal-current.cases* intro: *SAT.intros*)

lemma *orthogonal-currentD-A*: $\llbracket orthogonal-current \Gamma f S; x \in S; x \in A \Gamma; x \notin B \Gamma \rrbracket \implies d-OUT f x = weight \Gamma x$

by(auto elim: *orthogonal-current.cases*)

lemma *orthogonal-currentD-in*: $\llbracket orthogonal-current \Gamma f S; v \in RF_\Gamma S; u \notin RF^\circ_\Gamma S \rrbracket \implies f(u, v) = 0$

by(auto elim: *orthogonal-current.cases*)

inductive *linkage* :: ('v, 'more) web-scheme $\Rightarrow$ 'v current $\Rightarrow$ bool

for Γf

where — Omit the condition *web-flow*

linkage: $(\bigwedge x. x \in A \Gamma \implies d-OUT f x = weight \Gamma x) \implies linkage \Gamma f$

lemma *linkageD*: $\llbracket linkage \Gamma f; x \in A \Gamma \rrbracket \implies d-OUT f x = weight \Gamma x$

by(rule *linkage.cases*)

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

where $linkable \Gamma \equiv \exists f. web-flow \Gamma f \wedge linkage \Gamma f$

6.6 Trimming

context

fixes $\Gamma :: ('v, 'more) web-scheme$ (**structure**)
 and $f :: 'v current$

begin

inductive $trimming :: 'v current \Rightarrow bool$

for g

where

$trimming$:

— omits the condition that f is a wave

$\llbracket current \Gamma g; wave \Gamma g; g \leq f; \bigwedge x. \llbracket x \in RF^\circ (TER f); x \notin A \Gamma \rrbracket \Longrightarrow KIR g$
 $x; \mathcal{E} (TER g) - A \Gamma = \mathcal{E} (TER f) - A \Gamma \rrbracket$
 $\Longrightarrow trimming g$

lemma *assumes* $trimming g$

shows $trimmingD-current: current \Gamma g$

and $trimmingD-wave: wave \Gamma g$

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 $f: current \Gamma f$

and $w: wave \Gamma f$

and *countable*: **countable** $\mathbf{E}$

and *weight-finite*: $\bigwedge x. weight \Gamma x \neq \top$

shows $\exists g. trimming g$

proof —

define F **where** $F = \{g. current \Gamma g \wedge wave \Gamma g \wedge g \leq f \wedge \mathcal{E} (TER g) = \mathcal{E} (TER f)\}$

define leq **where** $leq = restrict-rel F \{(g, g'). g' \leq g\}$

have $in-F [simp]: g \in F \longleftrightarrow current \Gamma g \wedge wave \Gamma g \wedge (\forall e. g e \leq f e) \wedge \mathcal{E} (TER g) = \mathcal{E} (TER f)$ **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** $\dots \leq weight \Gamma y$ **by**(*rule currentD-weight-IN[OF f]*)

also **have** $\dots < \top$ **by**(*simp add: weight-finite less-top[symmetric]*)

finally **show** *?thesis* **using** $Pair$ **by** $simp$

qed

have *chainD*: $\text{Inf } M \in F$ **if** $M: M \in \text{Chains } \text{leq}$ **and** *nempty*: $M \neq \{\}$ **for** M
proof –
from *nempty* **obtain** $g0$ **where** $g0: g0 \in M$ **by** *auto*
have $g0\text{-le-}f$: $g0 \leq f \ e$ **and** g : *current* $\Gamma \ g0$ **and** $w0$: *wave* $\Gamma \ g0$ **for** e
using *Chains-FieldD*[$OF \ M \ g0$] **by**(*cases* e , *auto simp add: leq-def*)

have *finite-OUT*: $d\text{-OUT } f \ x \neq \top$ **for** x **using** *weight-finite*[$of \ x$]
by(*rule neq-top-trans*)(*rule currentD-weight-OUT*[$OF \ f$])
have *finite-IN*: $d\text{-IN } f \ x \neq \top$ **for** x **using** *weight-finite*[$of \ x$]
by(*rule neq-top-trans*)(*rule currentD-weight-IN*[$OF \ f$])

from M **have** $M \in \text{Chains } \{(g, g'). g' \leq g\}$
by(*rule mono-Chains*[*THEN subsetD, rotated*])(*auto simp add: leq-def in-restrict-rel-iff*)
then have *chain*: *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 *countable'*: *countable* (*support-flow* f)
using *current-support-flow*[$OF \ f$] **by**(*rule countable-subset*)(*rule countable*)

have *OUT-M*: $d\text{-OUT } (\text{Inf } M) \ x = (\text{INF } g \in M. d\text{-OUT } g \ x)$ **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*: $d\text{-IN } (\text{Inf } M) \ x = (\text{INF } g \in M. d\text{-IN } g \ x)$ **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* $\Gamma \ (\text{Inf } M)$ **using** g
proof(*rule current-leI*)
show $(\text{Inf } M) \ e \leq g0 \ e$ **for** e **using** $g0$ **by**(*auto intro: INF-lower*)
show $d\text{-OUT } (\bigsqcap M) \ x \leq d\text{-IN } (\bigsqcap M) \ x$ **if** $x \notin A \ \Gamma$ **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*[$OF \ - \ \text{that}$])
qed

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

have w' : *wave* $\Gamma \ (\text{Inf } M)$

```

proof
  have separating  $\Gamma$  ( $\mathcal{E}$  ( $TER$   $f$ )) by(rule separating-essential)(rule waveD-separating[ $OF$ 
 $w$ ])
    then show separating  $\Gamma$  ( $TER$  ( $Inf$   $M$ )) 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 ( $\lambda(y, x). (if\ x = z\ then\ factor\ else\ 1) * g\ (y, x)$ ) 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  $\Gamma\ g$  and  $w'$ : wave  $\Gamma\ g$  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  $\Gamma\ TER\ g$ ] 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:  $d-OUT\ g\ z \neq d-IN\ g\ z$ 
    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 ≤ 1 (is ?factor ≤ -)
  by (auto intro!: divide-le-posI-ennreal simp: zero-less-iff-neq-zero)
hence le': ?factor * g (y, x) ≤ 1 * g (y, x) for y x
  by (rule mult-right-mono) simp
hence le: trim1 g e ≤ 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 ∉ A Γ
    have d-OUT (trim1 g) x ≤ d-OUT g x unfolding d-OUT-def using le'
  by (auto intro: nn-integral-mono)
    also have ... ≤ 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 ≤ d-IN g x using x by (rule currentD-OUT-IN[OF g])
    also have ... ≤ 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 ≤ d-IN (trim1 g) x .
  qed
  moreover have le-f: trim1 g ≤ f using le le-f by (blast intro: le-funI or-
der-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 ∈  $\mathcal{E} (TER g)$ 
    hence x ∈ 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 ≠ 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 ∈  $\mathcal{E} (TER g)$ ⟩ have x ∈ SAT Γ (trim1 g) by (auto simp add:
SAT.simps)
    ultimately show x ∈ TER (trim1 g) by auto
  qed
  moreover have wave Γ (trim1 g)

```

```

proof
  have separating  $\Gamma$  ( $\mathcal{E}$  ( $TER$   $f$ )) by(rule separating-essential)(rule waveD-separating[ $OF$ 
 $w$ ])
    then show separating  $\Gamma$  ( $TER$  ( $trim1$   $g$ )) 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:  $g \leq f$ 
  and g: current  $\Gamma$   $g$ 
  and w': wave  $\Gamma$   $g$ 
  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 =  $\lambda x. x \in RF^\circ$  ( $TER$   $g$ )  $\wedge x \notin A$   $\Gamma \wedge \neg KIR$   $g$   $x$ 
    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  $\Gamma$   $TER$   $g$ , symmetric]
roofed-circ-essential[of  $\Gamma$   $TER$   $f$ , symmetric])
    hence ?P  $x$  unfolding x-def by(rule someI-ex)
    hence  $x: x \in RF^\circ$  ( $TER$   $g$ ) and  $A: x \notin A$   $\Gamma$  and neq:  $d-OUT$   $g$   $x \neq d-IN$   $g$   $x$ 
  by simp-all
  from neq have  $\exists y. edge$   $\Gamma$   $y$   $x \wedge g$  ( $y$ ,  $x$ )  $> 0$ 
  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  $\Gamma$  y x and gr0:  $g(y, x) > 0$  by blast

  have [simp]:  $g(y, x) \neq \top$ 
  proof -
    have  $g(y, x) \leq d\text{-}OUT\ g\ y$  by (rule d-OUT-ge-point)
    also have  $\dots \leq weight\ \Gamma\ y$  by (rule currentD-weight-OUT[OF g])
    also have  $\dots < \top$  by (simp add: weight-finite less-top[symmetric])
    finally show ?thesis by simp
  qed

  from neg have factor:  $d\text{-}OUT\ g\ x / d\text{-}IN\ g\ x \neq 1$ 
    by (simp add: divide-eq-1-ennreal)

  have  $trim1\ g\ (y, x) = g(y, x) * (d\text{-}OUT\ g\ x / d\text{-}IN\ g\ x)$ 
    by (clarsimp simp add: False trim1-def Let-def x-def[symmetric] mult.commute)
  moreover have  $\dots \neq g(y, x) * 1$  unfolding ennreal-mult-cancel-left using
gr0 factor by auto
  ultimately have  $trim1\ g\ (y, x) \neq g(y, x)$  by auto
  hence  $trim1\ g \neq g$  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-E:
  fixes  $\Gamma$  (structure)
  assumes w: wave  $\Gamma\ f$  and trimming: trimming  $\Gamma\ f\ g$ 
  shows  $\mathcal{E}\ (TER\ f) = \mathcal{E}\ (TER\ g)$ 
  proof (rule set-eqI)
    show  $x \in \mathcal{E}\ (TER\ f) \longleftrightarrow x \in \mathcal{E}\ (TER\ g)$  for x
    proof (cases  $x \in A\ \Gamma$ )
      case False
      thus ?thesis using trimmingD-E[OF trimming] by blast
    next
      case True
      show ?thesis
      proof
        assume x:  $x \in \mathcal{E}\ (TER\ f)$ 
        hence  $x \in TER\ g$  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  $\Gamma\ (B\ \Gamma)\ (TER\ f)\ x$  by simp
        then obtain p y where p: path  $\Gamma\ x\ p\ y$  and y:  $y \in B\ \Gamma$ 
          and bypass:  $\bigwedge z. z \in set\ p \implies z \notin RF\ (TER\ f)$  by (rule essentialE-RF)
      end
    end
  end

```

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

6.7 Composition of waves via quotients

definition $\text{quotient-web} :: ('v, 'more) \text{web-scheme} \Rightarrow 'v \text{ current} \Rightarrow ('v, 'more) \text{web-scheme}$

where — Modifications to original Definition 4.9: No incoming edges to nodes in A , $B \ \Gamma - A \ \Gamma$ is not part of A such that A contains only vertices is disjoint from B . The weight of vertices in B saturated by f is therefore set to 0.

$\text{quotient-web } \Gamma \ f =$
 $(\text{edge} = \lambda x \ y. \text{edge } \Gamma \ x \ y \wedge x \notin \text{roofed-circ } \Gamma \ (TER_{\Gamma} \ f) \wedge y \notin \text{roofed } \Gamma \ (TER_{\Gamma} \ f)),$
 $\text{weight} = \lambda x. \text{if } x \in RF^{\circ}_{\Gamma} \ (TER_{\Gamma} \ f) \vee x \in TER_{\Gamma} \ f \cap B \ \Gamma \text{ then } 0 \text{ else weight } \Gamma \ x,$
 $A = \mathcal{E}_{\Gamma} \ (TER_{\Gamma} \ f) - (B \ \Gamma - A \ \Gamma),$
 $B = B \ \Gamma,$

... = web.more Γ)

lemma *quotient-web-sel* [simp]:

fixes Γ (**structure**) **shows**

$\text{edge } (\text{quotient-web } \Gamma f) x y \longleftrightarrow \text{edge } \Gamma x y \wedge x \notin RF^\circ (TER f) \wedge y \notin RF (TER f)$

$\text{weight } (\text{quotient-web } \Gamma f) x = (\text{if } x \in RF^\circ (TER f) \vee x \in TER_\Gamma f \cap B \Gamma \text{ then } 0 \text{ else } \text{weight } \Gamma x)$

$A (\text{quotient-web } \Gamma f) = \mathcal{E} (TER f) - (B \Gamma - A \Gamma)$

$B (\text{quotient-web } \Gamma f) = B \Gamma$

$\text{web.more } (\text{quotient-web } \Gamma f) = \text{web.more } \Gamma$

by(simp-all add: quotient-web-def)

lemma *vertex-quotient-webD*: **fixes** Γ (**structure**) **shows**

$\text{vertex } (\text{quotient-web } \Gamma f) x \implies \text{vertex } \Gamma x \wedge x \notin RF^\circ (TER f)$

by(auto simp add: vertex-def roofed-circ-def)

lemma *path-quotient-web*:

fixes Γ (**structure**)

assumes $\text{path } \Gamma x p y$

and $x \notin RF^\circ (TER f)$

and $\bigwedge z. z \in \text{set } p \implies z \notin RF (TER f)$

shows $\text{path } (\text{quotient-web } \Gamma f) x p y$

using *assms* **by**(induction)(auto intro: rtrancl-path.intros simp add: roofed-circ-def)

definition *restrict-current* :: $('v, 'more) \text{ web-scheme} \Rightarrow 'v \text{ current} \Rightarrow 'v \text{ current} \Rightarrow 'v \text{ current}$

where $\text{restrict-current } \Gamma f g = (\lambda(x, y). g(x, y) * \text{indicator } (- RF^\circ_\Gamma (TER_\Gamma f)) x * \text{indicator } (- RF_\Gamma (TER_\Gamma f)) y)$

abbreviation *restrict-curr* :: $'v \text{ current} \Rightarrow ('v, 'more) \text{ web-scheme} \Rightarrow 'v \text{ current} \Rightarrow 'v \text{ current} (\langle - \rangle - ' / - \rangle [100, 0, 100] 100)$

where $\text{restrict-curr } g \Gamma f \equiv \text{restrict-current } \Gamma f g$

lemma *restrict-current-simps* [simp]: **fixes** Γ (**structure**) **shows**

$(g \upharpoonright \Gamma / f)(x, y) = (g(x, y) * \text{indicator } (- RF^\circ (TER f)) x * \text{indicator } (- RF (TER f)) y)$

by(simp add: restrict-current-def)

lemma *d-OUT-restrict-current-outside*: **fixes** Γ (**structure**) **shows**

$x \in RF^\circ (TER f) \implies d\text{-OUT } (g \upharpoonright \Gamma / f) x = 0$

by(simp add: d-OUT-def)

lemma *d-IN-restrict-current-outside*: **fixes** Γ (**structure**) **shows**

$x \in RF (TER f) \implies d\text{-IN } (g \upharpoonright \Gamma / f) x = 0$

by(simp add: d-IN-def)

lemma *restrict-current-le*: $(g \upharpoonright \Gamma / f) e \leq g e$

by(cases e)(clarsimp split: split-indicator)

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

lemma *d-IN-restrict-current-le*: $d\text{-IN } (g \upharpoonright \Gamma / f) x \leq d\text{-IN } g x$
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 $d\text{-IN } (g \upharpoonright \Gamma / f) x = d\text{-IN } g x$
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 : $z \in B \Gamma$ **and** p : *path* Γ x p z
 and *bypass*: $\bigwedge z. z \in \text{set } p \implies z \notin TER f \ x \notin TER f$
 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* $x \in TER f \Gamma$] 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) $\implies d\text{-IN } (g \upharpoonright \Gamma / f) a = 0$
by(*simp add: d-IN-restrict-current-outside roofed-greaterI*)

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

lemma *in-SINK-restrict-current*: $x \in SINK g \implies x \in SINK (g \upharpoonright \Gamma / f)$
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  $\Gamma$   $f$ 
and  $g$ : current  $\Gamma$   $g$ 
shows  $SAT$  (quotient-web  $\Gamma$   $f$ ) ( $g \upharpoonright \Gamma / f$ ) =  $RF$  ( $TER$   $f$ )  $\cup$  ( $SAT$   $\Gamma$   $g$  -  $A$   $\Gamma$ )
(is  $SAT$   $?T$   $?g$  =  $?rhs$ )
proof(intro set-eqI iffI; (elim UnE DiffE)?)
  show  $x \in ?rhs$  if  $x \in SAT$   $?T$   $?g$  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  $x \in SAT$   $?T$   $?g$  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  $x \in SAT$   $?T$   $?g$  if  $x \in SAT$   $\Gamma$   $g$   $x \notin A$   $\Gamma$  for  $x$  using that
      by(auto simp add: SAT.simps roofed-circ-def d-IN-restrict-current-outside re-
strict-current-IN-not-RF[ $OF$   $g$ ])
    qed

lemma current-restrict-current:
  fixes  $\Gamma$  (structure)
  assumes  $w$ : wave  $\Gamma$   $f$ 
  and  $g$ : current  $\Gamma$   $g$ 
  shows current (quotient-web  $\Gamma$   $f$ ) ( $g \upharpoonright \Gamma / f$ ) (is current  $?T$   $?g$ )
proof
  show  $d-OUT$   $?g$   $x \leq weight$   $?T$   $x$  for  $x$ 
    using d-OUT-restrict-current-le[of  $\Gamma$   $f$   $g$   $x$ ] currentD-weight-OUT[ $OF$   $g$ , of  $x$ ]
currentD-OUT[ $OF$   $g$ , of  $x$ ]
    by(auto simp add: d-OUT-restrict-current-outside)
  show  $d-IN$   $?g$   $x \leq weight$   $?T$   $x$  for  $x$ 
    using d-IN-restrict-current-le[of  $\Gamma$   $f$   $g$   $x$ ] 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  $x$   $\Gamma$   $f$   $g$ ]; simp add: roofed-greaterI)

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

```

```

    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 ≤ d-OUT g x
    by(rule d-OUT-mono restrict-current-le)+
  moreover have x ∉ 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 ∈ A  $\text{?}\Gamma$  for a using that by(rule restrict-current-IN-A)
show d-OUT ?g b = 0 if b ∈ B  $\text{?}\Gamma$  for b
  using d-OUT-restrict-current-le[of  $\Gamma$  f g b] currentD-OUT[OF g, of b] that by
simp
show ?g e = 0 if e ∉ E  $\text{?}\Gamma$  for e using that currentD-outside'[OF g, of e]
  by(cases e)(auto split: split-indicator)
qed

lemma TER-restrict-current:
  fixes  $\Gamma$  (structure)
  assumes f: current  $\Gamma$  f
  and w: wave  $\Gamma$  f
  and g: current  $\Gamma$  g
  shows TER g ⊆ TERquotient-web  $\Gamma$  f (g  $\upharpoonright$   $\Gamma$  / f) (is - ⊆ ?TER is - ⊆ TER  $\text{?}\Gamma$ 
?g)
proof
  fix x
  assume x: x ∈ TER g
  hence x ∈ SINK ?g by(simp add: in-SINK-restrict-current)
  moreover have x ∈ RF (TER f) if x ∈ A  $\Gamma$ 
    using waveD-separating[OF w, THEN separatingD, OF - that] by(rule roofedI)
  then have x ∈ SAT ? $\Gamma$  ?g using SAT-restrict-current[OF f g] x by auto
  ultimately show x ∈ ?TER by simp
qed

lemma wave-restrict-current:
  fixes  $\Gamma$  (structure)
  assumes f: current  $\Gamma$  f
  and w: wave  $\Gamma$  f
  and g: current  $\Gamma$  g
  and w': wave  $\Gamma$  g
  shows wave (quotient-web  $\Gamma$  f) (g  $\upharpoonright$   $\Gamma$  / f) (is wave ? $\Gamma$  ?g)
proof
  show separating ? $\Gamma$  (TER? $\Gamma$  ?g) (is separating - ?TER)
  proof
    fix x y p
    assume x ∈ A  $\text{?}\Gamma$  y ∈ B  $\text{?}\Gamma$  and p: path ? $\Gamma$  x p y
    hence x: x ∈  $\mathcal{E}$  (TER f) and y: y ∈ B  $\Gamma$  and SAT: x ∈ SAT ? $\Gamma$  ?g by(simp-all
add: SAT.A)

```

```

from  $p$  have  $p'$ :  $\text{path } \Gamma \ x \ p \ y$  by( $\text{rule } \text{rtrancl-path-mono}$ )  $\text{simp}$ 

{ assume  $x \notin ?TER$ 
  hence  $x \notin \text{SINK } ?g$  using  $\text{SAT}$  by( $\text{simp}$ )
  hence  $x \notin \text{SINK } g$  using  $d\text{-OUT-restrict-current-le}[of \ \Gamma \ f \ g \ x]$ 
    by( $\text{auto simp add: SINK.simps}$ )
  hence  $x \in RF \ (TER \ g)$  using  $\text{waveD-OUT}[OF \ w']$  by( $\text{auto simp add:}$ 
 $\text{SINK.simps}$ )
  from  $\text{roofedD}[OF \ \text{this } p' \ y] \ \langle x \notin \text{SINK } g \rangle$  have  $(\exists z \in \text{set } p. z \in ?TER)$ 
    using  $\text{TER-restrict-current}[OF \ f \ w \ g]$  by  $\text{blast}$  }
then show  $(\exists z \in \text{set } p. z \in ?TER) \vee x \in ?TER$  by  $\text{blast}$ 
qed

fix  $x$ 
assume  $x \notin RF_{?T} \ ?TER$ 
hence  $x \notin RF \ (TER \ g)$ 
proof( $\text{rule } \text{contrapos-nn}$ )
  assume *:  $x \in RF \ (TER \ g)$ 
  show  $x \in RF_{?T} \ ?TER$ 
proof
  fix  $p \ y$ 
  assume  $\text{path } ?T \ x \ p \ y \ y \in B \ ?T$ 
  hence  $\text{path } \Gamma \ x \ p \ y \ y \in B \ \Gamma$  by( $\text{auto elim: } \text{rtrancl-path-mono}$ )
  from  $\text{roofedD}[OF \ * \ \text{this}]$  show  $(\exists z \in \text{set } p. z \in ?TER) \vee x \in ?TER$ 
    using  $\text{TER-restrict-current}[OF \ f \ w \ g]$  by  $\text{blast}$ 
qed
qed
with  $w'$  have  $d\text{-OUT } g \ x = 0$  by( $\text{rule } \text{waveD-OUT}$ )
with  $d\text{-OUT-restrict-current-le}[of \ \Gamma \ f \ g \ x]$ 
show  $d\text{-OUT } ?g \ x = 0$  by  $\text{simp}$ 
qed

definition  $\text{plus-current} :: 'v \ \text{current} \Rightarrow 'v \ \text{current} \Rightarrow 'v \ \text{current}$ 
where  $\text{plus-current } f \ g = (\lambda e. f \ e + g \ e)$ 

lemma  $\text{plus-current-simps}$  [ $\text{simp}$ ]:  $\text{plus-current } f \ g \ e = f \ e + g \ e$ 
by( $\text{simp add: plus-current-def}$ )

lemma  $\text{plus-zero-current}$  [ $\text{simp}$ ]:  $\text{plus-current } f \ \text{zero-current} = f$ 
by( $\text{simp add: fun-eq-iff}$ )

lemma  $\text{support-flow-plus-current}$ :  $\text{support-flow } (\text{plus-current } f \ g) \subseteq \text{support-flow } f$ 
 $\cup \text{support-flow } g$ 
by( $\text{clarsimp simp add: support-flow.simps}$ )

context
fixes  $\Gamma :: ('v, 'more) \ \text{web-scheme} \ (\text{structure})$  and  $f \ g$ 
assumes  $f$ :  $\text{current } \Gamma \ f$ 
and  $w$ :  $\text{wave } \Gamma \ f$ 

```

and g : *current* (*quotient-web* Γ f) g
begin

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

proof –

have $d\text{-OUT } ?g \ x = d\text{-OUT } f \ x + d\text{-OUT } g \ x$ **unfolding** *plus-current-def*
by(*subst d-OUT-add*) *simp-all*
also have $\dots = (\text{if } x \in RF^\circ \ (TER \ f) \text{ then } d\text{-OUT } f \ x \text{ else } d\text{-OUT } g \ x)$
proof(*cases* $x \in RF^\circ \ (TER \ f)$)
case *True*
hence $d\text{-OUT } g \ x = 0$ **by**(*intro currentD-outside-OUT[OF g]*)(*auto dest: vertex-quotient-webD*)
thus *?thesis* **using** *True* **by** *simp*
next
case *False*
hence $d\text{-OUT } f \ x = 0$ **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*: $d\text{-IN } (\text{plus-current } f \ g) \ x = (\text{if } x \in RF \ (TER \ f) \text{ then } d\text{-IN } f \ x \text{ else } d\text{-IN } g \ x) \text{ (is } d\text{-IN } ?g \ - \ -)$

proof –

have $d\text{-IN } ?g \ x = d\text{-IN } f \ x + d\text{-IN } g \ x$ **unfolding** *plus-current-def*
by(*subst d-IN-add*) *simp-all*
also consider $(RF) \ x \in RF \ (TER \ f) - (B \ \Gamma - A \ \Gamma) \mid (B) \ x \in RF \ (TER \ f) \ x \in B \ \Gamma - A \ \Gamma \mid (\text{beyond}) \ x \notin RF \ (TER \ f)$ **by** *blast*
then have $d\text{-IN } f \ x + d\text{-IN } g \ x = (\text{if } x \in RF \ (TER \ f) \text{ then } d\text{-IN } f \ x \text{ else } d\text{-IN } g \ x)$
proof(*cases*)
case *RF*
hence $d\text{-IN } g \ x = 0$
by(*cases* $x \in \mathcal{E} \ (TER \ f)$)(*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 $d\text{-IN } g \ x = 0$ **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 $f \ w \text{ beyond}$ **have** $d\text{-IN } f \ x = 0$ **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 $x: x \in TER_{quotient-web}\ \Gamma\ f\ g$ (**is** $- \in ?TER\ -$)
shows $x \in TER\ (plus-current\ f\ g)$ (**is** $- \in TER\ ?g$)
proof(*cases* $x \in \mathcal{E}\ (TER\ f) - (B\ \Gamma - A\ \Gamma)$)
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 $x \in SAT\ \Gamma\ ?g$
proof(*cases* $x \in B\ \Gamma - A\ \Gamma$)
case *False*
with $x\ RF\ *$ **have** $weight\ \Gamma\ x \leq d-IN\ g\ x$
by(*auto elim!*: *SAT.cases split: if-split-asm simp add: essential-BI*)
also have $\dots \leq d-IN\ ?g\ x$ **unfolding** *plus-current-def* **by**(*intro d-IN-mono*)
simp
finally show ?thesis ..
next
case *True*
with $*$ x **have** $weight\ \Gamma\ x \leq d-IN\ ?g\ x$ **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*: $current\ \Gamma\ (plus-current\ f\ g)$ (**is** $current\ -\ ?g$)
proof
show $d-OUT\ ?g\ x \leq weight\ \Gamma\ x$ **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 $d-IN\ ?g\ x \leq weight\ \Gamma\ x$ **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: or-*
der-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-OUT f x = 0 weight  $\Gamma$  x  $\leq$  d-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-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-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  $e \notin E$  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  $\Gamma$  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 ( $\exists z \in \text{set } p. z \in TER$  (plus-current f g))  $\vee$   $x \in TER$  (plus-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  $\exists z \in \text{set } (x \# p). z \in \mathcal{E}$  (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:  $\bigwedge z. z \in \text{set } zs \implies z \notin \mathcal{E}$  (TER f) by auto
  have zs: path  $\Gamma$  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  $z' \in \text{set } zs$  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  $\Gamma$  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' ∈ set (z # zs) and TER: z' ∈ TERquotient-web Γ f g
  by auto
    hence z' ∈ TER ?g using in-TER-plus-current[of z'] RF[of z'] ‹z ∉ RFo (TER
f)› by(auto simp add: roofed-circ-def)
    with decomp z' show ?thesis by(cases ys) auto
  next
    case True
    hence z ∈ 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?Γ

  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 ∉ RF (TER ?g)
  hence x ∉ RF (TER f) by(rule contrapos-nn)(rule roofedI, rule separating-TER-plus-current)
  hence *: x ∉ RFo (TER f) by(simp add: roofed-circ-def)
  moreover have x ∉ RF?Γ (?TER g)
  proof
    assume RF': x ∈ RF?Γ (?TER g)
    from x obtain p y where y: y ∈ B Γ and p: path Γ x p y
      and bypass:  $\bigwedge z. z \in \text{set } p \implies z \notin \text{TER } ?g$  and x': x ∉ TER ?g
      by(auto simp add: roofed-def)
    have RF: z ∉ RF (TER f) if z ∈ set p for z
    proof
      assume z: z ∈ 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 ∈ ?TER g | (z) z where z ∈ set
p z ∈ ?TER g by auto
    then show False
  proof(cases)
    case x
    with * have x ∈ 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  $d-OUT\ g\ x = 0$  by (rule waveD-OUT)
ultimately show  $d-OUT\ ?g\ x = 0$  by (simp add: OUT-plus-current)
qed

end

end

lemma loose-quotient-web:
  fixes  $\Gamma :: ('v, 'more)\ web-scheme\ (\mathbf{structure})$ 
  assumes weight-finite:  $\bigwedge x. weight\ \Gamma\ x \neq \top$ 
  and f: current  $\Gamma\ f$ 
  and w: wave  $\Gamma\ f$ 
  and maximal:  $\bigwedge w. [\text{current}\ \Gamma\ w; \text{wave}\ \Gamma\ w; f \leq w] \implies f = w$ 
  shows loose (quotient-web  $\Gamma\ f$ ) (is loose  $? \Gamma$ )
proof
  fix  $g$ 
  assume g: current  $? \Gamma\ g$  and  $w'$ : wave  $? \Gamma\ g$ 
  let  $?g = plus-current\ f\ g$ 
  from  $f\ w\ g$  have current  $\Gamma\ ?g$  wave  $\Gamma\ ?g$  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  $f\ e \leq d-OUT\ f\ x$  unfolding Pair by (rule d-OUT-ge-point)
    also have  $\dots \leq weight\ \Gamma\ x$  by (rule currentD-weight-OUT[OF f])
    also have  $\dots < \top$  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  $g = (\lambda \cdot. 0)$  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  $\mathcal{E}: x \notin \mathcal{E}_{? \Gamma}\ (TER_{? \Gamma}\ zero-current)$ 

```


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

lemma quotient-web-trimming:
 fixes Γ (**structure**)
 assumes $w: \text{wave } \Gamma \ f$
 and $\text{trimming}: \text{trimming } \Gamma \ f \ g$
 shows $\text{quotient-web } \Gamma \ f = \text{quotient-web } \Gamma \ g$ (**is $?lhs = ?rhs$**)
proof (rule web.equality)
 from trimming have $\mathcal{E}: \mathcal{E} \ (TER \ g) - A \ \Gamma = \mathcal{E} \ (TER \ f) - A \ \Gamma$ **by cases**

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

```

show edge ?lhs = edge ?rhs by(rule ext)+(simp add: RF RFc)
have weight ?lhs = ( $\lambda x. \text{if } x \in RF^\circ \text{ (TER } g) \vee x \in RF \text{ (TER } g) \cap B \Gamma \text{ then } 0$ 
else weight  $\Gamma x$ )
  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  $\Gamma :: ('v, 'more) \text{ web-scheme (structure)}$ 
  assumes A-in:  $x \in A \Gamma \implies \neg \text{edge } \Gamma y x$ 
  and B-out:  $x \in B \Gamma \implies \neg \text{edge } \Gamma x y$ 
  and A-vertex:  $A \Gamma \subseteq \mathbf{V}$ 
  and disjoint:  $A \Gamma \cap B \Gamma = \{\}$ 
  and no-loop:  $\bigwedge x. \neg \text{edge } \Gamma x x$ 
  and weight-outside:  $\bigwedge x. x \notin \mathbf{V} \implies \text{weight } \Gamma x = 0$ 
  and weight-finite [simp]:  $\bigwedge x. \text{weight } \Gamma x \neq \top$ 
begin

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

lemma currentI [intro?]:
  assumes  $\bigwedge x. d\text{-OUT } f x \leq \text{weight } \Gamma x$ 
  and  $\bigwedge x. d\text{-IN } f x \leq \text{weight } \Gamma x$ 
  and OUT-IN:  $\bigwedge x. \llbracket x \notin A \Gamma; x \notin B \Gamma \rrbracket \implies d\text{-OUT } f x \leq d\text{-IN } f x$ 
  and outside:  $\bigwedge e. e \notin \mathbf{E} \implies f e = 0$ 
  shows current  $\Gamma f$ 
proof
  show  $d\text{-IN } f a = 0$  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  $d\text{-OUT } f b = 0$  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  $d\text{-OUT } f x \leq d\text{-IN } f x$  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  $\Gamma f$ 

```

```

  shows  $d\text{-IN } f x \neq \top$ 
proof(cases  $x \in \mathbf{V}$ )
  case True
    have  $d\text{-IN } f x \leq \text{weight } \Gamma x$  using  $f$  by(rule currentD-weight-IN)
    also have  $\dots < \top$  using True weight-finite[of  $x$ ] by (simp add: less-top[symmetric])
    finally show ?thesis by simp
  next
    case False
    then have  $d\text{-IN } f x = 0$ 
      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  $\Gamma f$ 
  shows  $d\text{-OUT } f x \neq \top$ 
proof(cases  $x \in \mathbf{V}$ )
  case True
    have  $d\text{-OUT } f x \leq \text{weight } \Gamma x$  using  $f$  by(rule currentD-weight-OUT)
    also have  $\dots < \top$  using True weight-finite[of  $x$ ] by (simp add: less-top[symmetric])
    finally show ?thesis by simp
  next
    case False
    then have  $d\text{-OUT } f x = 0$ 
      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  $\Gamma f$ 
  shows  $f e \neq \top$ 
proof(cases  $e$ )
  case (Pair  $x y$ )
    have  $f(x, y) \leq d\text{-OUT } f x$  by (rule d-OUT-ge-point)
    also have  $\dots < \top$  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  $\Gamma f$ ) (is web ? $\Gamma$ )
proof
  show  $\neg \text{edge } ?\Gamma y x$  if  $x \in A$  ? $\Gamma$  for  $x y$  using that by(auto intro: roofed-greaterI)
  show  $\neg \text{edge } ?\Gamma x y$  if  $x \in B$  ? $\Gamma$  for  $x y$  using that by(auto simp add: B-out)
  show  $A ?\Gamma \cap B ?\Gamma = \{\}$  using disjoint by auto
  show  $A ?\Gamma \subseteq \mathbf{V}_{?\Gamma}$ 
proof
  fix  $x$ 
  assume  $x \in A$  ? $\Gamma$ 

```

```

    hence  $\mathcal{E}: x \in \mathcal{E} \text{ (TER } f)$  and  $x: x \notin B \Gamma$  using disjoint by auto
    from this(1) obtain  $p \ y$  where  $p: \text{path } \Gamma \ x \ p \ y$  and  $y: y \in B \Gamma$  and bypass:
 $\bigwedge z. z \in \text{set } p \implies z \notin RF \text{ (TER } f)$ 
    by(rule  $\mathcal{E}$ -E-RF) blast
    from  $p \ y \ x$  have  $p \neq []$  by(auto simp add: rtrancl-path-simps)
    with rtrancl-path-nth[OF  $p$ , of 0] have  $\text{edge } \Gamma \ x \ (p ! 0)$  by simp
    moreover have  $x \notin RF^\circ \text{ (TER } f)$  using  $\mathcal{E}$  by(simp add: roofed-circ-def)
    moreover have  $p ! 0 \notin RF \text{ (TER } f)$  using bypass  $\langle p \neq [] \rangle$  by auto
    ultimately have  $\text{edge } ?\Gamma \ x \ (p ! 0)$  by simp
    thus  $x \in V_{?\Gamma}$  by(auto intro: vertexI1)
  qed
  show  $\neg \text{edge } ?\Gamma \ x \ x$  for  $x$  by(simp add: no-loop)
  show  $\text{weight } ?\Gamma \ x = 0$  if  $x \notin V_{?\Gamma}$  for  $x$ 
  proof(cases  $x \in RF^\circ \text{ (TER } f) \vee x \in \text{TER } f \cap B \Gamma$ )
    case True thus ?thesis by simp
  next
    case False
    hence  $RF: x \notin RF^\circ \text{ (TER } f)$  and  $B: x \in B \Gamma \implies x \notin \text{TER } f$  by auto
    from  $RF$  obtain  $p \ y$  where  $p: \text{path } \Gamma \ x \ p \ y$  and  $y: y \in B \Gamma$  and bypass:  $\bigwedge z. z \in \text{set } p \implies z \notin RF \text{ (TER } f)$ 
    apply(cases  $x \notin RF \text{ (RF (TER } f))$ )
    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  $x \notin \text{TER } f$  by simp
    hence  $RF': x \notin RF \text{ (TER } f)$  using  $xy \ y$  by(subst RF-in-B) auto
    have  $\neg \text{vertex } \Gamma \ x$ 
    proof
      assume vertex  $\Gamma \ x$ 
      then obtain  $x'$  where  $\text{edge } \Gamma \ x' \ x$  using  $xy \ y$  by(auto simp add: vertex-def
B-out)
      moreover hence  $x' \notin RF^\circ \text{ (TER } f)$  using  $RF'$  by(auto dest: RF-circ-edge-forward)
      ultimately have  $\text{edge } ?\Gamma \ x' \ x$  using  $RF'$  by simp
      hence  $\text{vertex } ?\Gamma \ x$  by(rule vertexI2)
      with that show False by simp
    qed
    thus ?thesis by(simp add: weight-outside)
  qed
  show  $\text{weight } ?\Gamma \ x \neq \top$  for  $x$  by simp
qed

end

locale countable-web = web  $\Gamma$ 
  for  $\Gamma :: ('v, 'more) \text{web-scheme}$  (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  $\Gamma$   $f$ ) (is countable-web ? $\Gamma$ )
proof –
  interpret  $r$ : web ? $\Gamma$  by(rule web-quotient-web)
  show ?thesis
  proof
    have  $\mathbf{E}_{\text{?}\Gamma} \subseteq \mathbf{E}$  by auto
    then show countable  $\mathbf{E}_{\text{?}\Gamma}$  by(rule countable-subset) simp
  qed
qed

end

```

6.9 Subtraction of a wave

definition $\text{minus-web} :: ('v, 'more) \text{web-scheme} \Rightarrow 'v \text{ current} \Rightarrow ('v, 'more) \text{web-scheme}$
(infixl $\ominus$ **65)** — Definition 6.6
where $\Gamma \ominus f = \Gamma \langle \text{weight} := \lambda x. \text{if } x \in A \ \Gamma \text{ then } \text{weight } \Gamma \ x - d\text{-OUT } f \ x \text{ else } \text{weight } \Gamma \ x + d\text{-OUT } f \ x - d\text{-IN } f \ x \rangle$

```

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

```

```

lemma vertex-minus-web [simp]: vertex ( $\Gamma \ominus f$ ) = vertex  $\Gamma$ 
by(simp add: vertex-def fun-eq-iff)

```

```

lemma roofed-gen-minus-web [simp]: roofed-gen ( $\Gamma \ominus f$ ) = roofed-gen  $\Gamma$ 
by(simp add: fun-eq-iff roofed-def)

```

```

lemma minus-zero-current [simp]:  $\Gamma \ominus \text{zero-current} = \Gamma$ 
by(rule web.equality)(simp-all add: fun-eq-iff)

```

```

lemma (in web) web-minus-web:
  assumes  $f$ : current  $\Gamma \ f$ 
  shows web ( $\Gamma \ominus f$ )

```

```

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] cur-
rentD-finite-OUT[OF f])
done

```

6.10 Bipartite webs

```

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

```

```

lemma not-vertex:  $\llbracket x \notin A \ \Gamma; x \notin B \ \Gamma \rrbracket \implies \neg \text{vertex } \Gamma \ x$ 
using bipartite-V by blast

```

```

lemma no-loop:  $\neg \text{edge } \Gamma \ x \ x$ 
using disjoint by (auto dest: bipartite-E)

```

```

lemma edge-antiparallel:  $\text{edge } \Gamma \ x \ y \implies \neg \text{edge } \Gamma \ y \ x$ 
using disjoint by (auto dest: bipartite-E)

```

```

lemma A-in:  $x \in A \ \Gamma \implies \neg \text{edge } \Gamma \ y \ x$ 
using disjoint by (auto dest: bipartite-E)

```

```

lemma B-out:  $x \in B \ \Gamma \implies \neg \text{edge } \Gamma \ x \ y$ 
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  $\Gamma \ f$ 
  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  $\Gamma \ f$ 
  and x:  $x \notin B \ \Gamma$ 

```

```

    shows  $d\text{-IN } f x = 0$ 
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:  $\bigwedge x. d\text{-OUT } f x \leq \text{weight } \Gamma x$ 
  and IN:  $\bigwedge x. d\text{-IN } f x \leq \text{weight } \Gamma x$ 
  and outside:  $\bigwedge e. e \notin \mathbf{E} \implies f e = 0$ 
  shows current  $\Gamma f$ 
proof
  fix x
  assume  $x \notin A \ \Gamma \ x \notin B \ \Gamma$ 
  hence  $d\text{-OUT } f x = 0$  by(auto simp add: d-OUT-def nn-integral-0-iff emea-
sure-count-space-eq-0 intro!: outside dest: bipartite-E)
  then show  $d\text{-OUT } f x \leq d\text{-IN } f x$  by simp
qed(rule OUT IN outside)+

lemma wave-bipartiteI [intro?]:
  assumes sep: separating  $\Gamma$  (TER f)
  and f: current  $\Gamma f$ 
  shows wave  $\Gamma f$ 
using sep
proof(rule wave.intros)
  fix x
  assume  $x \notin RF$  (TER f)
  then consider  $x \notin \mathbf{V} \mid x \in \mathbf{V} \ x \in B \ \Gamma$  using separating-RF-A[OF sep] bipartite-V
by auto
  then show  $d\text{-OUT } f x = 0$  using currentD-OUT[OF f, of x] currentD-outside-OUT[OF
f, of x]
    by cases auto
qed

lemma web-flow-iff: web-flow  $\Gamma f \longleftrightarrow \text{current } \Gamma f$ 
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 ::  $('v, 'more) \text{ web-scheme} \Rightarrow ('v + 'v, 'more) \text{ web-scheme}$ 

```

where

$\text{bipartite-web-of } \Gamma =$
 $\langle \text{edge} = \lambda uv. \text{ case } (uv, uv') \text{ of } (Inl\ u, Inr\ v) \Rightarrow \text{edge } \Gamma\ u\ v \vee u = v \wedge u \in$
 $\text{vertices } \Gamma \wedge u \notin A\ \Gamma \wedge v \notin B\ \Gamma \mid - \Rightarrow \text{False},$
 $\text{weight} = \lambda uv. \text{ case } uv \text{ of } Inl\ u \Rightarrow \text{if } u \in B\ \Gamma \text{ then } 0 \text{ else weight } \Gamma\ u \mid Inr\ u \Rightarrow$
 $\text{if } u \in A\ \Gamma \text{ then } 0 \text{ else weight } \Gamma\ u,$
 $A = Inl\ ' (vertices\ \Gamma - B\ \Gamma),$
 $B = Inr\ ' (- A\ \Gamma),$
 $\dots = \text{web.more } \Gamma \rangle$

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

$\text{edge } (\text{bipartite-web-of } \Gamma) (Inl\ u) (Inr\ v) \longleftrightarrow \text{edge } \Gamma\ u\ v \vee u = v \wedge u \in \mathbf{V} \wedge u$
 $\notin A\ \Gamma \wedge v \notin B\ \Gamma$
 $\text{edge } (\text{bipartite-web-of } \Gamma) uv (Inl\ u) \longleftrightarrow \text{False}$
 $\text{edge } (\text{bipartite-web-of } \Gamma) (Inr\ v) uv \longleftrightarrow \text{False}$
 $\text{weight } (\text{bipartite-web-of } \Gamma) (Inl\ u) = (\text{if } u \in B\ \Gamma \text{ then } 0 \text{ else weight } \Gamma\ u)$
 $\text{weight } (\text{bipartite-web-of } \Gamma) (Inr\ v) = (\text{if } v \in A\ \Gamma \text{ then } 0 \text{ else weight } \Gamma\ v)$
 $A\ (\text{bipartite-web-of } \Gamma) = Inl\ ' (\mathbf{V} - B\ \Gamma)$
 $B\ (\text{bipartite-web-of } \Gamma) = Inr\ ' (- A\ \Gamma)$
by(simp-all add: bipartite-web-of-def split: sum.split)

lemma *edge-bipartite-webI1*: $\text{edge } \Gamma\ u\ v \implies \text{edge } (\text{bipartite-web-of } \Gamma) (Inl\ u) (Inr\ v)$
by(auto)

lemma *edge-bipartite-webI2*:

$\llbracket u \in \mathbf{V}_\Gamma; u \notin A\ \Gamma; u \notin B\ \Gamma \rrbracket \implies \text{edge } (\text{bipartite-web-of } \Gamma) (Inl\ u) (Inr\ u)$
by(auto)

lemma *edge-bipartite-webE*:

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

context *web* **begin**

lemma *vertex-bipartite-web* [simp]:

$\text{vertex } (\text{bipartite-web-of } \Gamma) (Inl\ x) \longleftrightarrow \text{vertex } \Gamma\ x \wedge x \notin B\ \Gamma$
 $\text{vertex } (\text{bipartite-web-of } \Gamma) (Inr\ x) \longleftrightarrow \text{vertex } \Gamma\ x \wedge x \notin A\ \Gamma$

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 $\cap$ B-S) $\cup$ (A Γ $\cap$ A-S) $\cup$ (B Γ $\cap$ B-S))

context

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

Proof of separation follows [1]

lemma *separating-of-bipartite-aux*:

assumes p: path Γ x p y **and** y: y $\in$ B Γ
and x: x $\in$ A $\Gamma \vee$ Inr x $\in$ S
shows ($\exists z \in \text{set } p. z \in \text{separating-of-bipartite } S$) $\vee$ x $\in$ *separating-of-bipartite* S
proof(cases p = [])
case True
with p **have** x = y **by** cases auto
with y x **have** x $\in$ *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 $\longleftrightarrow$ x $\in$ A $\Gamma \vee$ Inr x $\in$ S **for** x
define outmarked **where** outmarked x $\longleftrightarrow$ x $\in$ B $\Gamma \vee$ Inl x $\in$ S **for** x
let ? Γ = bipartite-web-of Γ
let ?double = $\lambda x. \text{inmarked } x \wedge \text{outmarked } x$
define tailmarked **where** tailmarked = ($\lambda(x, y :: 'v). \text{outmarked } x$)
define headmarked **where** headmarked = ($\lambda(x :: 'v, y). \text{inmarked } y$)

have marked-E: tailmarked e $\vee$ headmarked e **if** e $\in$ **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 $\exists z \in \text{set } (x \# p). ?double z$ — Lemma 2

proof —

have inmarked ((x # p) ! (i + 1)) $\vee$ 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 $\wedge$ inmarked (p ! i)} $\cup$ {i. i < length (x # butlast p) $\wedge$ 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 ((λi. 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 in-
marked-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 $f e \leq d\text{-OUT } f x$ **unfolding** *Pair d-OUT-def* **by**(*rule nn-integral-ge-point*)
simp
also have $\dots \leq \text{weight } ?\Gamma x$ **by**(*rule currentD-weight-OUT[OF f]*)
also have $\dots < \top$ **by**(*cases x*)(*simp-all add: less-top[symmetric]*)
finally show *?thesis* **by** *simp*
qed

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

lemma *current-of-bipartite-simps* [*simp*]: *current-of-bipartite* $f (x, y) = f (Inl x, Inr y) * \text{indicator } \mathbf{E} (x, y)$
by(*simp add: current-of-bipartite-def*)

lemma *d-OUT-current-of-bipartite*:
assumes f : *current* (*bipartite-web-of* Γ) f
shows $d\text{-OUT } (\text{current-of-bipartite } f) x = d\text{-OUT } f (Inl x) - f (Inl x, Inr x)$
proof –
have $d\text{-OUT } (\text{current-of-bipartite } f) x = \int^+ y. f (Inl x, y) * \text{indicator } \mathbf{E} (x, \text{projr } y) \partial \text{count-space } (\text{range } Inr)$
by(*simp add: d-OUT-def nn-integral-count-space-reindex*)
also have $\dots = d\text{-OUT } f (Inl x) - \int^+ y. f (Inl x, y) * \text{indicator } \{Inr x\} y \partial \text{count-space } UNIV$ (**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 $d\text{-IN } (\text{current-of-bipartite } f) x = d\text{-IN } f (Inr x) - f (Inl x, Inr x)$
proof –
have $d\text{-IN } (\text{current-of-bipartite } f) x = \int^+ y. f (y, Inr x) * \text{indicator } \mathbf{E} (\text{projl } y, x) \partial \text{count-space } (\text{range } Inl)$
by(*simp add: d-IN-def nn-integral-count-space-reindex*)
also have $\dots = d\text{-IN } f (Inr x) - \int^+ y. f (y, Inr x) * \text{indicator } \{Inl x\} y \partial \text{count-space } UNIV$ (**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  $\Gamma$ )  $f$  (is current  $?T$  -)
and  $w$ : wave (bipartite-web-of  $\Gamma$ )  $f$ 
shows current  $\Gamma$  (current-of-bipartite  $f$ ) (is current -  $?f$ )
proof
  fix  $x$ 
  have  $d\text{-OUT } ?f\ x \leq d\text{-OUT } f\ (Inl\ x)$ 
    by(simp add: d-OUT-current-of-bipartite[ $OF\ f$ ] diff-le-self-ennreal)
  also have  $\dots \leq \text{weight } \Gamma\ x$ 
    using currentD-weight-OUT[ $OF\ f$ , of Inl x]
    by(simp split: if-split-asm)
  finally show  $d\text{-OUT } ?f\ x \leq \text{weight } \Gamma\ x$  .
next
  fix  $x$ 
  have  $d\text{-IN } ?f\ x \leq d\text{-IN } f\ (Inr\ x)$ 
    by(simp add: d-IN-current-of-bipartite[ $OF\ f$ ] diff-le-self-ennreal)
  also have  $\dots \leq \text{weight } \Gamma\ x$ 
    using currentD-weight-IN[ $OF\ f$ , of Inr x]
    by(simp split: if-split-asm)
  finally show  $d\text{-IN } ?f\ x \leq \text{weight } \Gamma\ x$  .
next
  have  $OUT: d\text{-OUT } ?f\ b = 0$  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 intro!:
currentD-outside[ $OF\ f$ ] dest: B-out)
  show  $d\text{-OUT } ?f\ x \leq d\text{-IN } ?f\ x$  if  $A: x \notin A\ \Gamma$  for  $x$ 
  proof(cases  $x \in B\ \Gamma \vee x \notin V$ )
    case True
    then show  $?thesis$ 
    proof
      assume  $x \in B\ \Gamma$ 
      with  $OUT$ [ $OF\ this$ ] show  $?thesis$  by auto
    next
      assume  $x \notin V$ 
      hence  $d\text{-OUT } ?f\ x = 0$  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  $d\text{-OUT } ?f\ x = d\text{-OUT } f\ (Inl\ x) - f\ (Inl\ x, Inr\ x)$  (is  $- = - - ?rest$ )
      by(simp add: d-OUT-current-of-bipartite[ $OF\ f$ ])
    also have  $d\text{-OUT } f\ (Inl\ x) \leq d\text{-IN } f\ (Inr\ x)$ 
      proof(rule ccontr)
        assume  $\neg ?thesis$ 
        hence  $∗: d\text{-IN } f\ (Inr\ x) < d\text{-OUT } f\ (Inl\ x)$  by(simp add: not-less)
        also have  $\dots \leq \text{weight } \Gamma\ x$  using currentD-weight-OUT[ $OF\ f$ , of Inl x]  $B$ 
      by simp
    finally have  $Inr\ x \notin TER_{\mathcal{T}}\ f$  using  $A$  by(auto elim!: SAT.cases)
    moreover have  $Inl\ x \notin TER_{\mathcal{T}}\ f$  using  $∗$  by(auto simp add: SINK.simps)
    moreover have  $path\ ?T\ (Inl\ x)\ [Inr\ x]\ (Inr\ x)$ 

```

$\text{by}(\text{rule } r\text{tranc1-path.step})(\text{auto intro!}: r\text{tranc1-path.base simp add: no-loop } A$
 $B)$
 $\text{ultimately show False using waveD-separating}[OF w] A B \text{ by}(\text{auto dest!}: \text{separatingD})$
 qed
 $\text{hence } d\text{-OUT } f (Inl x) - ?rest \leq d\text{-IN } f (Inr x) - ?rest \text{ by}(\text{rule ennreal-minus-mono})$
 simp
 $\text{also have } \dots = d\text{-IN } ?f x \text{ by}(\text{simp add: } d\text{-IN-current-of-bipartite}[OF f])$
 $\text{finally show } ?thesis .$
 qed
 $\text{show } ?f e = 0 \text{ if } e \notin E \text{ for } e \text{ using that by}(\text{cases } e)(\text{auto})$
 qed

lemma *TER-current-of-bipartite*: — Lemma 6.3
 $\text{assumes } f: \text{current } (bipartite\text{-web-of } \Gamma) f \text{ (is current } ?T -)$
 $\text{and } w: \text{wave } (bipartite\text{-web-of } \Gamma) f$
 $\text{shows } TER (\text{current-of-bipartite } f) = \text{separating-of-bipartite } (TER_{bipartite\text{-web-of } \Gamma} f)$
 $(\text{is } TER ?f = \text{separating-of-bipartite } ?TER)$
proof(*rule set-eqI*)
 $\text{fix } x$
 $\text{consider } (A) x \in A \Gamma x \in \mathbf{V} \mid (B) x \in B \Gamma x \in \mathbf{V}$
 $\mid (inner) x \notin A \Gamma x \notin B \Gamma x \in \mathbf{V} \mid (outside) x \notin \mathbf{V} \text{ by auto}$
 $\text{thus } x \in TER ?f \longleftrightarrow x \in \text{separating-of-bipartite } ?TER$
proof *cases*
 $\text{case } A$
 $\text{hence } d\text{-OUT } ?f x = d\text{-OUT } f (Inl x) \text{ using currentD-outside}[OF f, \text{ of } Inl x$
 $Inr x]$
 $\text{by}(\text{simp add: } d\text{-OUT-current-of-bipartite}[OF f] \text{ no-loop})$
 $\text{thus } ?thesis \text{ using } A \text{ disjoint}$
 $\text{by}(\text{auto simp add: separating-of-bipartite-def Let-def SINK.simps intro!}: SAT.A$
 $\text{imageI})$
 next
 $\text{case } B$
 $\text{then have } d\text{-IN } ?f x = d\text{-IN } f (Inr x)$
 $\text{using currentD-outside}[OF f, \text{ of } Inl x Inr x]$
 $\text{by}(\text{simp add: } d\text{-IN-current-of-bipartite}[OF f] \text{ no-loop})$
 $\text{moreover have } d\text{-OUT } ?f x = 0 \text{ using } B \text{ currentD-outside}[OF f, \text{ of } Inl x Inr$
 $x]$
 $\text{by}(\text{simp add: } d\text{-OUT-current-of-bipartite}[OF f] \text{ no-loop})(\text{auto simp add: } d\text{-OUT-def nn-integral-0-iff emeasure-count-space-eq-0 intro!}: \text{currentD-outside}[OF$
 $f] \text{ elim!}: \text{edge-bipartite-webE dest: B-out})$
 $\text{moreover have } d\text{-OUT } f (Inr x) = 0 \text{ using } B \text{ disjoint by}(\text{intro currentD-OUT}[OF$
 $f]) \text{ auto}$
 $\text{ultimately show } ?thesis \text{ using } B$
 $\text{by}(\text{auto simp add: separating-of-bipartite-def Let-def SINK.simps SAT.simps})$
 next
 case outside
 $\text{with current-current-of-bipartite}[OF f w] \text{ have } d\text{-OUT } ?f x = 0 \text{ } d\text{-IN } ?f x = 0$

```

    by(rule currentD-outside-OUT currentD-outside-IN)+
    moreover from outside have Inl x  $\notin$  vertices ?T Inr x  $\notin$  vertices ?T 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  $\in$  separating-of-bipartite ?TER
      with inner have l: Inl x  $\in$  ?TER and r: Inr x  $\in$  ?TER
      by(auto simp add: separating-of-bipartite-def Let-def)
      have f (Inl x, Inr x)  $\leq$  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  $\in$  SINK ?f by(simp add: SINK.simps d-OUT-current-of-bipartite[OF
f])
      moreover from r have Inr x  $\in$  SAT ?T f by auto
      with inner have x  $\in$  SAT  $\Gamma$  ?f
      by(auto elim!: SAT.cases intro!: SAT.IN simp add: 0 d-IN-current-of-bipartite[OF
f])
      ultimately show x  $\in$  TER ?f by simp
    next
      assume *: x  $\in$  TER ?f
      have d-IN f (Inr x)  $\leq$  weight ?T (Inr x) using f by(rule currentD-weight-IN)
      also have ...  $\leq$  weight  $\Gamma$  x using inner by simp
      also have ...  $\leq$  d-IN ?f x using inner * by(auto elim: SAT.cases)
      also have ...  $\leq$  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] cur-
rentD-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  $\in$  ?TER Inr x  $\in$  ?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  $\in$  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  $\Gamma$ )  $f$  (is current ? $\Gamma$  -)
  and  $w$ : wave (bipartite-web-of  $\Gamma$ )  $f$ 
  shows wave  $\Gamma$  (current-of-bipartite  $f$ ) (is wave - ? $f$ )
proof
  have  $sep'$ : separating  $\Gamma$  (separating-of-bipartite (TER? $\Gamma$   $f$ ))
    by(rule separating-of-bipartite)(rule waveD-separating[OF  $w$ ])
  then show  $sep$ : separating  $\Gamma$  (TER (current-of-bipartite  $f$ ))
    by(simp add: TER-current-of-bipartite[OF  $f$   $w$ ])

  fix  $x$ 
  assume  $x \notin RF$  (TER ? $f$ )
  then obtain  $p$   $y$  where  $p$ : path  $\Gamma$   $x$   $p$   $y$  and  $y$ :  $y \in B$   $\Gamma$  and  $x$ :  $x \notin TER$  ? $f$ 
    and  $bypass$ :  $\bigwedge z. z \in set\ p \implies z \notin TER$  ? $f$ 
    by(auto simp add: roofed-def elim: rtrancl-path-distinct)
  from  $p$  obtain  $p'$  where  $p'$ : path  $\Gamma$   $x$   $p'$   $y$  and  $distinct$ : distinct ( $x \# p'$ )
    and  $subset$ : set  $p' \subseteq set\ p$  by(auto elim: rtrancl-path-distinct)
  consider (outside)  $x \notin \mathbf{V} \mid (A) \ x \in A \ \Gamma \mid (B) \ x \in B \ \Gamma \mid (inner) \ x \in \mathbf{V} \ x \notin A$ 
 $\Gamma \ x \notin B \ \Gamma$  by auto
  then show  $d\text{-OUT}$  ? $f$   $x = 0$ 
  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  $x \in TER$  ? $f$  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 ? $\Gamma$  (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  $x \in TER_{? \Gamma} f \mid (Inr) \ Inr \ x \in TER_{? \Gamma} f$  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* $? \Gamma$)

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* $? \Gamma \ x \ y$ **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* $? \Gamma \ x \neq \top$ **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 $\neg$ *hindered* (*bipartite-web-of* Γ)
proof
 assume *hindered* (*bipartite-web-of* Γ) (**is** *hindered* $? \Gamma$)
 then obtain f **where** f : *current* $? \Gamma \ f$ **and** w : *wave* $? \Gamma \ f$ **and** $hind$: *hindrance* $? \Gamma \ f$ **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* $f = \text{zero-current}$ **by**(*rule looseD-wave[OF loose]*)
 have *zero*: $f \ (\text{Inl } x, \text{Inr } y) = 0$ **if** $x \neq y$ **for** $x \ y$ **using** *that* $*[\text{THEN fun-cong, of } (x, y)]$
 by(*cases edge* $\Gamma \ x \ y$)(*auto intro: currentD-outside[OF f]*)

 have *OUT*: $d\text{-OUT } f \ (\text{Inl } x) = f \ (\text{Inl } x, \text{Inr } x)$ **for** x
 proof $-$
 have $d\text{-OUT } f \ (\text{Inl } x) = (\sum^+ y. f \ (\text{Inl } x, y) * \text{indicator } \{\text{Inr } x\} \ y)$ **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 ? $\Gamma$  and n $\mathcal{E}$ : a  $\notin$   $\mathcal{E}$  ? $\Gamma$  (TER ? $\Gamma$  f)
  and OUT-a: d-OUT f a < weight ? $\Gamma$  a by cases
from a obtain a' where a': a = Inl a' and v: vertex  $\Gamma$  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 ? $\Gamma$  (Inl a') (Inr a') using v b by simp
  hence p: path ? $\Gamma$  (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 ? $\Gamma$  (B ? $\Gamma$ ) ?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  $\Gamma$  a' using A by(auto simp add: currentD-SAT[OF f])
  hence d-OUT f (Inl a') = weight  $\Gamma$  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 ?T (B ?T) ?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 ?T (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.prem(3) bypass[of x] weight-outside[of x] have x: vertex Γ x
by(auto)
    from ⟨edge Γ x z⟩ have path ?T (Inl x) [Inr z] (Inr z) by(simp add:
rtrancL-path-simps)
  from separatingD[OF waveD-separating[OF w] this] step.prem(1) step.prem(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.prem(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.prem(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*: $a / b < (1::ennreal) \longleftrightarrow (0 < b \wedge a < b \vee b = 0 \wedge a = 0 \vee b = \text{top})$
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 $?T$)

and *countable*: countable $\mathbf{E}$

shows linkable Γ

proof $-$

from *link* **obtain** f **where** *wf*: web-flow $?T$ f **and** *link*: linkage $?T$ f **by** blast

from *wf* **have** f : current $?T$ f **by** (rule web-flowD-current)

define f' **where** $f' = \text{current-of-bipartite } f$

have *IN-le-OUT*: $d\text{-IN } f' x \leq d\text{-OUT } f' x$ **if** $x \notin B \Gamma$ **for** x

proof (cases $x \in \mathbf{V}$)

case *True*

have $d\text{-IN } f' x = d\text{-IN } f (\text{Inr } x) - f (\text{Inl } x, \text{Inr } x)$ (**is** $- = - - ?rest$)

by (simp add: f' -def $d\text{-IN-current-of-bipartite}[OF f]$)

also have $\dots \leq \text{weight } ?T (\text{Inr } x) - ?rest$

using *currentD-weight-IN*[$OF f$, of $\text{Inr } x$] **by** (rule ennreal-minus-mono) simp

also have $\dots \leq \text{weight } ?T (\text{Inl } x) - ?rest$ **using** that *ennreal-minus-mono*

by (auto)

also have $\text{weight } ?T (\text{Inl } x) = d\text{-OUT } f (\text{Inl } x)$ **using** that *linkageD*[$OF \text{ link}$, of $\text{Inl } x$] *True* **by** auto

also have $\dots - ?rest = d\text{-OUT } f' x$

by (simp add: f' -def $d\text{-OUT-current-of-bipartite}[OF f]$)

finally show $?thesis$.

next

case *False*

with *currentD-outside-OUT*[$OF f$, of $\text{Inl } x$] *currentD-outside-IN*[$OF f$, of $\text{Inr } x$]

show $?thesis$ **by** (simp add: f' -def $d\text{-IN-current-of-bipartite}[OF f]$ $d\text{-OUT-current-of-bipartite}[OF f]$)

qed

have *link*: linkage Γ f'

proof

show $d\text{-OUT } f' a = \text{weight } \Gamma a$ **if** $a \in A \Gamma$ **for** a

proof (cases $a \in \mathbf{V}$)

case *True*

from that **have** $a \notin B \Gamma$ **using** disjoint **by** auto

with that *True linkageD*[$OF \text{ link}$, of $\text{Inl } a$] *ennreal-minus-cancel-iff*[of $- - 0$] *currentD-outside*[$OF f$, of $\text{Inl } a$ $\text{Inr } a$]

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

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

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

```

```

have chain-Field: Inf M ∈ F if M: M ∈ Chains leq and empty: M ≠ {} for M
proof -
  from empty obtain g0 where g0-in-M: g0 ∈ M by auto
  with M have g0: g0 ∈ Field leq by (rule Chains-FieldD)

  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 Complete-Partial-Order.chain (≥) M by (rule Chains-into-chain)
  hence chain': Complete-Partial-Order.chain (≤) M by (simp add: chain-dual)

  have support-flow f' ⊆ 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 ≠ ⊤ for x using weight-finite[of x]
    by (rule neg-top-trans)(rule F-weight-OUT[OF f'])
  have finite-IN: d-IN f' x ≠ ⊤ for x using weight-finite[of x]
    by (rule neg-top-trans)(rule F-weight-IN[OF f'])
  have OUT-M: d-OUT (Inf M) x = (INF g ∈ M. d-OUT g x) for x using chain'
  empty 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 ∈ M. d-IN g x) for x using chain' empty
  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 ∈ F using g0 unfolding F[symmetric]
  proof (rule F-leI)
    show (Inf M) e ≤ g0 e for e using g0-in-M by (auto intro: INF-lower)
    show 0 ≤ (Inf M) e for e by (auto intro!: INF-greatest dest: F-nonneg dest!:
Chains-FieldD[OF M])
    show d-IN (Inf M) x ≤ d-OUT (Inf M) x if x ∉ 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 empty
    by (simp add: linkage.simps OUT-M F-link[THEN linkageD] Chains-FieldD[OF
M] cong: INF-cong)
  qed
qed

let ?P = λg z. z ∉ A Γ ∧ z ∉ B Γ ∧ d-OUT g z > d-IN g z

define link
  where link g =
    (if ∃ z. ?P g z then
      let z = SOME z. ?P g z; factor = d-IN g z / d-OUT g z
      in (λ(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 = fixp-above f'

have g: g ∈ 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 d-OUT g x ≤ weight Γ x for x using g by(rule F-weight-OUT)
show d-IN g x ≤ weight Γ x for x using g by(rule F-weight-IN)
show d-IN g x = 0 if x ∈ A Γ for x using g that by(rule F-IN)
show B: d-OUT g x = 0 if x ∈ B Γ for x using g that by(rule F-OUT)
show g e = 0 if e ∉ E for e using g that by(rule F-outside)

show KIR: KIR g x if A: x ∉ A Γ and B: x ∉ B Γ for x
proof(rule ccontr)
define z where z = Eps (?P g)
assume ¬ KIR g x
with F-IN-OUT[OF g B] have d-OUT g x > d-IN g x by simp
with A B have Ex: ∃ x. ?P g x by blast
then 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 ∃ y. edge Γ z y ∧ g (z, y) > 0
proof(rule ccontr)
assume ¬ ?thesis
hence d-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 link g (z, y) ≠ g (z, y) using Ex by(auto simp add: link-def z-def
Let-def)
hence link g ≠ 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  $d\text{-OUT } g \ x \leq d\text{-IN } g \ x$  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  $\Gamma :: ('v, 'more) \text{ web-scheme (structure) and } h \ g$ 
  defines  $k \equiv \text{plus-current } h \ g$ 
  assumes  $f$ :  $\text{current } \Gamma \ f$ 
  and  $w$ :  $\text{wave } \Gamma \ f$ 
  and  $wg$ :  $\text{web-flow (quotient-web } \Gamma \ f) \ g \text{ (is web-flow } ?\Gamma \ -)$ 
  and  $link$ :  $\text{linkage (quotient-web } \Gamma \ f) \ g$ 
  and  $trim$ :  $\text{trimming } \Gamma \ f \ h$ 
  shows  $\text{web-flow } \Gamma \ k$ 
  and  $\text{orthogonal-current } \Gamma \ k \ (\mathcal{E} \ (TER \ f))$ 
proof -
  from  $wg$  have  $g$ :  $\text{current } ?\Gamma \ g$  by (rule web-flowD-current)

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

  have  $eq$ :  $\text{quotient-web } \Gamma \ f = \text{quotient-web } \Gamma \ h$  using  $w \ trim$  by (rule quotient-web-trimming)

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

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

```

```

have  $k$ : current  $\Gamma$   $k$  unfolding  $k$ -def using  $h$   $w'$   $g$  unfolding  $eq$  by(rule current-plus-current)
then show  $wk$ : web-flow  $\Gamma$   $k$ 
proof(rule web-flow)
  fix  $x$ 
  assume  $x \in \mathbf{V}$  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]:  $d\text{-OUT } h \ x = 0$  and  $d\text{-IN } h \ x \geq \text{weight } \Gamma \ x$ 
        by(auto simp add: SINK.simps elim: SAT.cases)
      with currentD-weight-IN[OF  $h$ , of  $x$ ] have [simp]:  $d\text{-IN } h \ x = \text{weight } \Gamma \ x$  by
auto
      from linkageD[OF link, of  $x$ ] True currentD-IN[OF  $g$ , of  $x$ ]  $B$ 
      have  $d\text{-OUT } g \ x = \text{weight } \Gamma \ x$   $d\text{-IN } g \ x = 0$  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  $h\text{-le-}k$ :  $h \ e \leq k \ e$  for  $e$  unfolding  $k$ -def plus-current-def by(rule add-increasing2)
simp-all
hence  $SAT \ \Gamma \ h \subseteq SAT \ \Gamma \ k$  by(rule SAT-mono)
hence  $SAT: ?T \subseteq SAT \ \Gamma \ k$  using TER by(auto simp add: elim!: SAT.cases intro: SAT.intros)
show orthogonal-current  $\Gamma$   $k$  ? $T$ 
proof(rule orthogonal-current)
  show  $\text{weight } \Gamma \ x \leq d\text{-IN } k \ x$  if  $x \in ?T$   $x \notin A \Gamma$  for  $x$ 
    using subsetD[OF  $SAT$ , of  $x$ ] that by(auto simp add: currentD-SAT[OF  $k$ ])
  next
    fix  $x$ 
    assume  $x: x \in ?T$  and  $A: x \in A \Gamma$  and  $B: x \notin B \Gamma$ 
    with d-OUT-mono[of  $h \ x \ f$ , OF  $h\text{-le-}f$ ] have  $d\text{-OUT } h \ x = 0$  by(auto simp
add: SINK.simps)
    moreover from linkageD[OF link, of  $x$ ]  $x \ A$  have  $d\text{-OUT } g \ x = \text{weight } ?\Gamma \ x$ 
by simp
    ultimately show  $d\text{-OUT } k \ x = \text{weight } \Gamma \ x$  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  $v: v \in RF \ ?T$  and  $u: u \notin RF^\circ \ ?T$ 
      have  $h \ (u, v) \leq f \ (u, v)$  by(rule h-le-f)
      also have  $\dots \leq d\text{-OUT } f \ u$  unfolding d-OUT-def by(rule nn-integral-ge-point)
simp
      also have  $\dots = 0$  using  $u$  using RF-essential[of  $\Gamma \ TER \ f$ ]

```

```

    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  $\Gamma$  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:  $\bigwedge f. \llbracket \text{current } \Gamma f; \text{wave } \Gamma f; \text{loose } (\text{quotient-web } \Gamma f) \rrbracket$ 
   $\implies$  linkable (quotient-web  $\Gamma f$ )
  shows  $\exists f S. \text{web-flow } \Gamma f \wedge \text{separating } \Gamma S \wedge \text{orthogonal-current } \Gamma f S$ 
proof —
  from ex-maximal-wave[OF countable]
  obtain  $f$  where  $f$ : current  $\Gamma f$ 
    and  $w$ : wave  $\Gamma f$ 
    and maximal:  $\bigwedge w. \llbracket \text{current } \Gamma w; \text{wave } \Gamma w; f \leq w \rrbracket \implies f = w$  by blast
  from ex-trimming[OF  $f w$  countable weight-finite] obtain  $h$  where  $h$ : trimming
 $\Gamma f h ..$ 

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

  let  $?k = \text{plus-current } h g$ 
  have web-flow  $\Gamma ?k$  orthogonal-current  $\Gamma ?k$  ( $\mathcal{E}(\text{TER } f)$ )
    by(rule linkage-quotient-webD[OF  $f w wg$  link  $h$ ])+
  moreover have separating  $\Gamma (\mathcal{E}(\text{TER } f))$ 
    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* :: $('v, 'more) \text{ network-scheme } \Rightarrow ('v \text{ edge}, 'more) \text{ web-scheme }$
where

```

web-of-network  $\Delta =$ 
  ( $\text{edge} = \lambda(x, y) (y', z). y' = y \wedge \text{edge } \Delta x y \wedge \text{edge } \Delta y z,$ 
    $\text{weight} = \text{capacity } \Delta,$ 
    $A = \{(\text{source } \Delta, x) | x. \text{edge } \Delta (\text{source } \Delta) x\},$ 
    $B = \{(x, \text{sink } \Delta) | x. \text{edge } \Delta x (\text{sink } \Delta)\},$ 
    $\dots = \text{network.more } \Delta$ )

```

lemma *web-of-network-sel* [*simp*]:

fixes Δ (**structure**) **shows**

edge (*web-of-network* Δ) $e \ e' \longleftrightarrow e \in \mathbf{E} \wedge e' \in \mathbf{E} \wedge \text{snd } e = \text{fst } e'$

weight (*web-of-network* Δ) $e = \text{capacity } \Delta \ e$

A (*web-of-network* Δ) = $\{(source \ \Delta, x) | x. \text{edge } \Delta \ (source \ \Delta) \ x\}$

B (*web-of-network* Δ) = $\{(x, sink \ \Delta) | x. \text{edge } \Delta \ x \ (sink \ \Delta)\}$

by(*auto simp add: web-of-network-def split: prod.split*)

lemma *vertex-web-of-network* [*simp*]:

vertex (*web-of-network* Δ) $(x, y) \longleftrightarrow \text{edge } \Delta \ x \ y \wedge (\exists z. \text{edge } \Delta \ y \ z \vee \text{edge } \Delta \ z \ x)$

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

definition *flow-of-current* :: $('v, 'more) \text{ network-scheme} \Rightarrow 'v \text{ edge current} \Rightarrow 'v \text{ flow}$

where *flow-of-current* $\Delta \ f \ e = \max \ (d\text{-OUT } f \ e) \ (d\text{-IN } f \ e)$

lemma *flow-flow-of-current*:

fixes Δ (**structure**) **and** Γ

defines [*simp*]: $\Gamma \equiv \text{web-of-network } \Delta$

assumes *fw*: *web-flow* $\Gamma \ f$

shows *flow* $\Delta \ (flow\text{-of-current } \Delta \ f)$ (**is** *flow* - ?*f*)

proof

from *fw* **have** *f*: *current* $\Gamma \ f$ **and** *KIR*: $\bigwedge x. \llbracket x \notin A \ \Gamma; x \notin B \ \Gamma \rrbracket \implies \text{KIR } f \ x$

by(*auto 4 3 dest: web-flowD-current web-flowD-KIR*)

show ?*f* $e \leq \text{capacity } \Delta \ e$ **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*: $x \neq source \ \Delta \ x \neq sink \ \Delta$

have *d-OUT* ?*f* $x = (\sum^+ y. d\text{-IN } f \ (x, y))$ **unfolding** *d-OUT-def*

by(*simp add: flow-of-current-def max-absorb2 currentD-OUT-IN*[*OF f*] *x*)

also have $\dots = (\sum^+ y. \sum^+ e \in range \ (\lambda z. (z, x)). f \ (e, x, y))$

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 $\dots = (\sum^+ z \in UNIV. \sum^+ y. f \ ((z, x), x, y))$

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 $\dots = (\sum^+ z. \sum^+ e \in range \ (Pair \ x). f \ ((z, x), e))$

by(*simp add: nn-integral-count-space-reindex*)

also have $\dots = (\sum^+ z. d\text{-OUT } f \ (z, x))$

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 $\dots = (\sum^+ z \in \{z. \text{edge } \Delta \ z \ x\}. d\text{-OUT } f \ (z, x))$

by(*auto intro!: nn-integral-cong currentD-outside-OUT*[*OF f*] *simp add: nn-integral-count-space-indicator split: split-indicator*)

```

also have ... = ( $\sum^+ z \in \{z. \text{edge } \Delta z x\}. \max (d\text{-OUT } f (z, x)) (d\text{-IN } f (z, x))$ )
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  $z \in \text{space } (\text{count-space } \{z. \text{edge } \Delta z x\})$  for  $z$  using  $\text{currentD-IN}[OF f]$ 
that
  by(cases  $z = \text{source } \Delta$ )(simp-all add: max-absorb1  $\text{currentD-IN}[OF f]$  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  $\text{currentD-outside-OUT}[OF f]$   $\text{currentD-outside-IN}[OF f]$ 
  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  $\Delta$  (structure)
assumes  $\text{sep: separating-network } \Delta V$ 
and  $\text{sink: sink } \Delta \notin V$ 
shows  $\text{cut } \Delta (RF^N V)$ 

```

proof

```

show  $\text{source } \Delta \in RF^N V$  by(rule roofedI)(auto dest: separatingD[OF sep])
show  $\text{sink } \Delta \notin RF^N V$  using  $\text{sink}$  by(auto dest: roofedD[OF - rtrancl-path.base])

```

qed

context

```

fixes  $\Delta :: ('v, 'more) \text{network-scheme}$  and  $\Gamma$  (structure)
defines  $\Gamma\text{-def: } \Gamma \equiv \text{web-of-network } \Delta$ 

```

begin

lemma *separating-network-cut-of-sep*:

```

assumes  $\text{sep: separating } \Gamma S$ 
and  $\text{source-sink: source } \Delta \neq \text{sink } \Delta$ 
shows  $\text{separating-network } \Delta (\text{fst } \mathcal{E} S)$ 

```

proof

```

define  $s\ t$  where  $s = \text{source } \Delta$  and  $t = \text{sink } \Delta$ 
fix  $p$ 

```

```

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

```

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

lemma *separating-cut*:

```

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

show  $\text{sink } \Delta \notin \text{fst } ' \mathcal{E} \ S$ 
proof
  assume  $\text{sink } \Delta \in \text{fst } ' \mathcal{E} \ S$ 
  then obtain  $x$  where  $(\text{sink } \Delta, x) \in \mathcal{E} \ S$  by auto
  hence  $(\text{sink } \Delta, x) \in \mathbf{V}$  by( $\text{auto simp add: } \Gamma\text{-def dest!: essential-vertex}$ )
  then show False by( $\text{simp add: } \Gamma\text{-def sink-out}$ )
qed
qed

```

context

```

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

```

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*: $x \in (\text{cut-of-sep } S)$

and *y*: $y \notin (\text{cut-of-sep } S)$

shows $(x, y) \in S$

proof –

from *y* **obtain** *p* **where** *p*: *path* Δ *y p* (*sink* Δ) **and** *y'*: $y \notin \text{fst } \mathcal{E} S$

and *bypass*: $\bigwedge z. z \in \text{set } p \implies z \notin \text{fst } \mathcal{E} S$ **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** $x \in \text{fst } \mathcal{E} S$ **by**(*auto simp add:* Γ -def[symmetric])

then obtain *z* **where** *xz*: $(x, z) \in \mathcal{E} S$ **by** *auto*

then obtain *q b* **where** *q*: *path* Γ $(x, z) q b$ **and** *b*: $b \in B \Gamma$

and *distinct*: *distinct* $((x, z) \# q)$ **and** *bypass'*: $\bigwedge z. z \in \text{set } q \implies z \notin RF S$

by(*rule* $\mathcal{E}$ -E-RF) *blast*

define *p'* **where** $p' = y \# p$

hence $p' \neq []$ **by** *simp*

with *p'* **have** *path* Γ $(x, \text{hd } p') (\text{zip } p' (\text{tl } p')) (\text{last } (x \# \text{butlast } p'), \text{sink } \Delta)$

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 $(x, y) (\text{zip } (y \# p) p) (\text{last } (x \# \text{butlast } (y \# p)), \text{sink } \Delta)$

(*is path* - ?*y* ?*p* ?*t*)

by(*simp add: p'-def*)

have $(?y \# ?p) ! \text{length } p = ?t$ **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 $\text{length } p < \text{length } (?y \# ?p)$ **by** *simp*

ultimately have *t*: ?*t* $\in B \Gamma$ **using** *rtrancl-path-nth*[*OF p''*, of $\text{length } p - 1$] *e*

by(*cases p*)(*simp-all add: Γ -def split: if-split-asm*)

show *S*: $(x, y) \in S$

proof(*cases x = source* Δ)

case *True*

from *separatingD*[*OF separating-essential*, *OF sep p'' - t*] *e True*

consider $(z) z z'$ **where** $(z, z') \in \text{set } ?p$ $(z, z') \in \mathcal{E} S \mid (x, y) \in S$ **by**(*auto*)

```

simp add:  $\Gamma$ -def)
thus ?thesis
proof cases
case (z z)
hence  $z \in \text{set } p$   $z \in \text{fst } \mathcal{E} \ S$ 
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  $\exists e. \text{edge } \Gamma \ e \ (x, z) \wedge f \ (e, (x, z)) > 0$ 
proof(rule ccontr)
assume  $\neg ?thesis$ 
then have  $d\text{-IN } f \ (x, z) = 0$  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  $(x, z) \in S$  by auto
hence  $(x, z) \in \text{SAT } \Gamma \ f$  by(rule orthogonal-currentD-SAT[OF ortho])
with False have  $d\text{-IN } f \ (x, z) \geq \text{capacity } \Delta \ (x, z)$  by(auto simp add:
SAT.simps  $\Gamma$ -def)
ultimately have  $\neg \text{capacity } \Delta \ (x, z) > 0$  by auto
hence  $\neg \text{edge } \Delta \ x \ z$  using capacity-pos[of  $(x, z)$ ] by auto
moreover with  $q$  have  $b = (x, z)$  by cases(auto simp add:  $\Gamma$ -def)
with  $b$  have  $\text{edge } \Delta \ x \ z$  by(simp add:  $\Gamma$ -def)
ultimately show False by contradiction
qed
then obtain  $u$  where  $ux: \text{edge } \Delta \ u \ x$  and  $xz': \text{edge } \Delta \ x \ z$  and  $uxz: \text{edge } \Gamma$ 
 $(u, x) \ (x, z)$ 
and  $gt0: f \ ((u, x), (x, z)) > 0$  by(auto simp add:  $\Gamma$ -def)
have  $(u, x) \in \text{RF}^\circ \ S$  using orthogonal-currentD-in[OF ortho, of (x, z) (u, x)]
gt0 xz
by(fastforce intro: roofed-greaterI)
hence  $ux\text{-RF}: (u, x) \in \text{RF} \ (\mathcal{E} \ S)$  and  $ux\text{-}\mathcal{E}: (u, x) \notin \mathcal{E} \ S$  unfolding RF-essential
by(simp-all add: roofed-circ-def)

from  $ux \ e$  have  $\text{edge } \Gamma \ (u, x) \ (x, y)$  by(simp add:  $\Gamma$ -def)
hence  $\text{path } \Gamma \ (u, x) \ ((x, y) \# ?p) \ ?t$  using  $p''$  ..
from roofedD[OF  $ux\text{-RF}$  this  $t$ ]  $ux\text{-}\mathcal{E}$ 
consider  $(x, y) \in S \mid (z) \ z \ z'$  where  $(z, z') \in \text{set } ?p \ (z, z') \in \mathcal{E} \ S$  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:  $\text{source } \Delta \neq \text{sink } \Delta$ 
  and sink-out:  $\bigwedge x. \neg \text{edge } \Delta (\text{sink } \Delta) x$ 
  and no-direct-edge:  $\neg \text{edge } \Delta (\text{source } \Delta) (\text{sink } \Delta)$  — Otherwise,  $A$  and  $B$  of the
  web would not be disjoint.
  shows orthogonal  $\Delta$  (flow-of-current  $\Delta f$ ) (cut-of-sep  $S$ ) (is orthogonal -  $?f ?S$ )
proof
  fix  $x y$ 
  assume  $e$ : edge  $\Delta x y$  and  $x \in ?S$  and  $y \notin ?S$ 
  then have  $S$ :  $(x, y) \in S$  by(rule orthogonal-leave-RF)

  show  $?f(x, y) = \text{capacity } \Delta(x, y)$ 
  proof(cases  $x = \text{source } \Delta$ )
    case False
      with orthogonal-currentD-SAT[OF ortho S]
      have  $\text{weight } \Gamma(x, y) \leq d\text{-IN } f(x, y)$  by cases(simp-all add:  $\Gamma$ -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  $\Gamma$ -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  $\Gamma$ -def max-def)
    qed
  next
    from sep source-sink sink-out have cut:  $\text{cut } \Delta ?S$  by(rule separating-cut)

    fix  $x y$ 
    assume  $xy$ : edge  $\Delta x y$ 
    and  $x$ :  $x \notin ?S$ 
    and  $y$ :  $y \in ?S$ 
    from  $x$  obtain  $p$  where  $p$ : path  $\Delta x p (\text{sink } \Delta)$  and  $x'$ :  $x \notin \text{fst } ' \mathcal{E} S$ 
    and bypass:  $\bigwedge z. z \in \text{set } p \implies z \notin \text{fst } ' \mathcal{E} S$  by(auto simp add: roofed-def
    cut-of-sep-def)
    have source:  $x \neq \text{source } \Delta$ 
    proof
      assume  $x = \text{source } \Delta$ 
      have separating-network  $\Delta (\text{fst } ' \mathcal{E} S)$  using sep source-sink by(rule separating-network-cut-of-sep)
      from separatingD[OF this p]  $\langle x = \text{source } \Delta \rangle x$  show False
      by(auto dest: bypass intro: roofed-greaterI simp add: cut-of-sep-def)
    qed
    hence  $A$ :  $(x, y) \notin A \Gamma$  by(simp add:  $\Gamma$ -def)

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

```

```

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

```

```

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 ∈ 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-ℰ consider (X) (x, hd p) ∈ ℰ S
  | (z) z z' where (z, z') ∈ set (zip p (tl p)) (z, z') ∈ 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 ⇒ 'a set where cut' S ≡ Vertex - ' S

lemma cut-cut': cut Δ'' S ⇒ cut Δ (cut' S)

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

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

lemma *OUT-Edge*: $\mathbf{OUT}_{\Delta''}(\text{Edge } x \ y) = (\text{if edge } \Delta \ x \ y \text{ then } \{\text{Vertex } y\} \text{ else } \{\})$
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:  $\bigwedge x. \neg \text{edge } \Delta x \text{ (source } \Delta)$ 
  and sink-out:  $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$ 
  and undead:  $\bigwedge x y. \text{edge } \Delta x y \implies (\exists z. \text{edge } \Delta y z) \vee (\exists z. \text{edge } \Delta z x)$ 
  and source-sink:  $\neg \text{edge } \Delta (\text{source } \Delta) (\text{sink } \Delta)$ 
  and no-loop:  $\bigwedge x. \neg \text{edge } \Delta x x$ 
  shows countable-web (web-of-network  $\Delta$ ) (is countable-web ? $\Gamma$ )

```

proof

```

  show  $\neg \text{edge } ?\Gamma y x$  if  $x \in A$  ? $\Gamma$  for  $x y$  using that by(clarsimp simp add:
source-in)
  show  $\neg \text{edge } ?\Gamma x y$  if  $x \in B$  ? $\Gamma$  for  $x y$  using that by(clarsimp simp add:
sink-out)
  show  $A \text{ ?}\Gamma \subseteq V_{?\Gamma}$  by(auto 4 3 dest: undead)
  show  $A \text{ ?}\Gamma \cap B \text{ ?}\Gamma = \{\}$  using source-sink by auto
  show  $\neg \text{edge } ?\Gamma x x$  for  $x$  by(auto simp add: no-loop)
  show weight ? $\Gamma$   $x = 0$  if  $x \notin V_{?\Gamma}$  for  $x$  using that undead
    by(cases x)(auto intro!: capacity-outside)
  show weight ? $\Gamma$   $x \neq \top$  for  $x$  using capacity-finite[of x] by(cases x) simp
  have  $E_{?\Gamma} \subseteq E \times E$  by auto
  thus countable  $E_{?\Gamma}$  by(rule countable-subset)(simp)
qed

```

lemma max-flow-min-cut':

```

  assumes ex-orthogonal-current:  $\exists f S. \text{web-flow (web-of-network } \Delta) f \wedge \text{sepa-}$ 
rating (web-of-network  $\Delta$ )  $S \wedge \text{orthogonal-current (web-of-network } \Delta) f S$ 
  and source-in:  $\bigwedge x. \neg \text{edge } \Delta x \text{ (source } \Delta)$ 
  and sink-out:  $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$ 
  and undead:  $\bigwedge x y. \text{edge } \Delta x y \implies (\exists z. \text{edge } \Delta y z) \vee (\exists z. \text{edge } \Delta z x)$ 
  and source-sink:  $\neg \text{edge } \Delta (\text{source } \Delta) (\text{sink } \Delta)$ 
  and no-loop:  $\bigwedge x. \neg \text{edge } \Delta x x$ 
  and capacity-pos:  $\bigwedge e. e \in E \implies \text{capacity } \Delta e > 0$ 

```

shows $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$
proof –
 let $? \Gamma = \text{web-of-network } \Delta$
 from $\text{ex-orthogonal-current}$ **obtain** $f S$
 where f : $\text{web-flow } (\text{web-of-network } \Delta) f$
 and S : $\text{separating } (\text{web-of-network } \Delta) S$
 and ortho : $\text{orthogonal-current } (\text{web-of-network } \Delta) f S$ **by** blast+
 let $?f = \text{flow-of-current } \Delta f$ **and** $?S = \text{cut-of-sep } \Delta S$
 from f **have** $\text{flow } \Delta ?f$ **by** $(\text{rule flow-flow-of-current})$
 moreover **have** $\text{cut } \Delta ?S$ **using** $S \text{ source-neq-sink sink-out}$ **by** $(\text{rule separating-cut})$
 moreover **have** $\text{orthogonal } \Delta ?f ?S$ **using** $f \text{ ortho } S \text{ capacity-pos source-neq-sink sink-out source-sink}$
by $(\text{rule orthogonal-flow-of-current})$
 ultimately **show** $?thesis$ **by** blast
qed

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

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

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

lemma Δ''' -countable-network: $\text{countable-network } \Delta'''$

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

lemma $\text{flow-}\Delta'''$:

assumes f : $\text{flow } \Delta''' f$ **and** cut : $\text{cut } \Delta''' S$ **and** ortho : $\text{orthogonal } \Delta''' f S$
 shows $\text{flow } \Delta f \text{ cut } \Delta S \text{ orthogonal } \Delta f S$
proof –

```

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

```

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

8.1 Linkages in un hindered 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 $\forall X \subseteq A \Gamma. \text{finite } X \longrightarrow (\sum^+ x \in X. \text{weight } \Gamma x) \leq (\sum^+ y \in \mathbf{E} \text{ `` } X. \text{weight } \Gamma y)$
using *assms*
proof(*rule contrapos-np*)
assume $\neg$ *?thesis*
then obtain X **where** $X \in \{X. X \subseteq A \Gamma \wedge \text{finite } X \wedge (\sum^+ y \in \mathbf{E} \text{ `` } X. \text{weight } \Gamma y) < (\sum^+ x \in X. \text{weight } \Gamma x)\}$ (**is** $- \in \text{Collect ?P}$)
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: $(\sum^+ y \in \mathbf{E} \text{ `` } X. \text{weight } \Gamma y) < (\sum^+ x \in X. \text{weight } \Gamma x)$
and minimal: $\bigwedge X'. X' \subset X \implies (\sum^+ x \in X'. \text{weight } \Gamma x) \leq (\sum^+ y \in \mathbf{E} \text{ `` } X'. \text{weight } \Gamma y)$
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**
 $f x = (\text{if } x = xx \text{ then } (\sum^+ y \in \mathbf{E} \text{ `` } X. \text{weight } \Gamma y) - (\sum^+ x \in X - \{xx\}. \text{weight } \Gamma x) \text{ else if } x \in X \text{ then weight } \Gamma x \text{ else } 0)$ **for** x
define g **where**
 $g y = (\text{if } y \in \mathbf{E} \text{ `` } X \text{ then weight } \Gamma y \text{ else } 0)$ **for** y
define E' **where** $E' \equiv \mathbf{E} \cap X \times \text{UNIV}$
have $Xxx: X - \{xx\} \subset X$ **using** xx **by** *blast*
have E *[simp]:* $E' \text{ `` } X' = \mathbf{E} \text{ `` } X' \text{ if } X' \subseteq X \text{ for } X'$ **using** *that* **by**(*auto simp add: E'-def*)
have *in-E':* $(x, y) \in E' \longleftrightarrow x \in X \wedge (x, y) \in \mathbf{E} \text{ for } x y$ **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 $\dots = (\sum^+ x \in X - \{xx\}. \text{weight } \Gamma x) + ((\sum^+ y \in \mathbf{E} \text{ `` } X. \text{weight } \Gamma y) - (\sum^+ x \in X - \{xx\}. \text{weight } \Gamma x))$
by(*rule arg-cong2[where f=(+)](auto simp add: f-def xx nn-integral-count-space-indicator intro!: nn-integral-cong)*)
also have $\dots = (\sum^+ y \in \mathbf{E} \text{ `` } X. g y)$ **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:* $(\sum^+ x \in X. f x) = (\sum^+ y \in \mathbf{E} \text{ `` } X. g y)$.

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

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

have $(\sum^+ x \in \mathbf{E} \text{ `` } X'. \text{ weight } \Gamma x) \leq (\sum^+ x \in \mathbf{E} \text{ `` } X. \text{ weight } \Gamma x)$ **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 $f\,xx = (\sum^+ y \in \mathbf{E} \text{ `` } X. \text{ weight } \Gamma y) - (\sum^+ x \in X - \{xx\}. \text{ weight } \Gamma x)$ **by**
(*simp add: f-def*)
also have $\dots < (\sum^+ x \in X. \text{ weight } \Gamma x) - (\sum^+ x \in X - \{xx\}. \text{ weight } \Gamma x)$
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 $\dots = (\sum^+ x \in \{xx\}. \text{ weight } \Gamma x)$ **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 $\dots = \text{weight } \Gamma\,xx$ **by**(*simp add: nn-integral-count-space-indicator*)
finally have $f\,xx: f\,xx < \text{weight } \Gamma\,xx$.

have $le: (\sum^+ x \in X'. f\,x) \leq (\sum^+ y \in \mathbf{E} \text{ `` } X'. g\,y)$ **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 $(\sum^+ x \in X'. f\,x) = (\sum^+ x \in X' - \{xx\}. f\,x) + (\sum^+ x \in \{xx\}. f\,x * \text{indicator } X'\,xx)$
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 $\dots \leq (\sum^+ x \in X' - \{xx\}. f\,x) + (\sum^+ x \in \{xx\}. \text{weight } \Gamma\,x * \text{indicator } X'\,xx)$ **using** *fxx*
by(*intro add-mono*)(*auto split: split-indicator simp add: nn-integral-count-space-indicator*)
also have $\dots = (\sum^+ x \in X'. \text{weight } \Gamma\,x)$ **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 $\dots \leq (\sum^+ y \in \mathbf{E} \text{ `` } X'. \text{weight } \Gamma\,y)$ **by**(*rule minimal[OF X']*)
also have $\dots = (\sum^+ y \in \mathbf{E} \text{ `` } X'. g\,y)$ **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 $\mathbf{E} \text{ `` } X \subseteq \mathbf{V}$ **by**(*auto simp add: vertex-def*)
with *countable-V* **have** *countable (E `` X)* **by**(*blast intro: countable-subset*)
moreover have $E' \subseteq X \times \mathbf{E} \text{ `` } X$ **by**(*auto simp add: E'-def*)
ultimately obtain h' **where** $h'\text{-dom}: \bigwedge x\,y. 0 < h'\,x\,y \implies (x, y) \in E'$

```

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

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

define h where  $h = (\lambda(x, y). \text{if } x \in X \wedge \text{edge } \Gamma x y \text{ then } h' x y \text{ else } 0)$ 
have h-OUT:  $d\text{-OUT } h x = (\text{if } x \in X \text{ then } f x \text{ else } 0)$  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:  $d\text{-IN } h y = (\text{if } y \in E' \text{ then } \text{weight } \Gamma y \text{ else } 0)$  for  $y$  using h'-g[of
 $y, \text{symmetric}$ ]
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  $\Gamma h$ 
proof
show  $d\text{-OUT } h x \leq \text{weight } \Gamma x$  for  $x$  using fx by(auto simp add: h-OUT
f-def)
show  $d\text{-IN } h y \leq \text{weight } \Gamma y$  for  $y$  by(simp add: h-IN)
show  $h e = 0$  if  $e \notin E$  for  $e$  using that by(cases e)(auto simp add: h-def)
qed

have separating  $\Gamma (TER h)$ 
proof
fix  $x y p$ 
assume  $x: x \in A \Gamma$  and  $y: y \in B \Gamma$  and  $p: \text{path } \Gamma x p y$ 
then obtain [simp]:  $p = [y]$  and  $xy: (x, y) \in E$  using disjoint
by  $\neg(\text{erule } r\text{tranc1-path.cases; auto dest: bipartite-}E)+$ 
show  $(\exists z \in \text{set } p. z \in TER h) \vee x \in TER h$ 
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  $\exists z \in \text{set } p. z \in TER h$  by simp
qed
qed
then have  $w: \text{wave } \Gamma h$  using  $h ..$ 

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

```

```

    assume  $xx \in \mathcal{E} \text{ (TER } h)$ 
    then obtain  $p \ y$  where  $y: y \in B \ \Gamma$  and  $p: \text{path } \Gamma \ xx \ p \ y$ 
      and  $\text{bypass}: \bigwedge z. \llbracket xx \neq y; z \in \text{set } p \rrbracket \implies z = xx \vee z \notin \text{TER } h$ 
      by( $\text{rule } \mathcal{E}\text{-E}$ ) auto
    from  $p$  obtain  $[simp]: p = [y]$  and  $xy: \text{edge } \Gamma \ xx \ y$  and  $\text{neg}: xx \neq y$  using
    disjoint  $X\text{-A } xx \ y$ 
      by  $-(\text{erule } r\text{tranc1-path.cases}; \text{auto } \text{dest}: \text{bipartite-E})+$ 
    from  $\text{neg } \text{bypass}[of \ y]$  have  $y \notin \text{TER } h$  by  $\text{simp}$ 
    moreover from  $xy \ xx \ \text{currentD-OUT}[OF \ h \ y]$  have  $y \in \text{TER } h$ 
      by( $\text{auto } \text{simp } \text{add}: \text{SAT.simps } h\text{-IN } \text{SINK.simps}$ )
    ultimately show  $\text{False}$  by  $\text{contradiction}$ 
  qed
  moreover have  $d\text{-OUT } h \ xx < \text{weight } \Gamma \ xx$  using  $fxx \ xx$  by( $\text{simp } \text{add}: h\text{-OUT}$ )
  ultimately have  $\text{hindrance } \Gamma \ h \ ..$ 
  then show  $\text{hindered } \Gamma$  using  $h \ w \ ..$ 
qed

end

lemma  $nn\text{-integral-count-space-top-approx}$ :
  fixes  $f :: \text{nat} \Rightarrow \text{ennreal}$  and  $b :: \text{ennreal}$ 
  assumes  $nn\text{-integral } (\text{count-space } UNIV) \ f = \text{top}$ 
  and  $b < \text{top}$ 
  obtains  $n$  where  $b < \text{sum } f \ \{..<n\}$ 
  using  $\text{assms } \text{unfolding } nn\text{-integral-count-space-nat } \text{suminf-eq-SUP } \text{SUP-eq-top-iff}$ 
  by( $\text{auto}$ )

lemma  $\text{One-le-of-nat-ennreal}$ :  $(1 :: \text{ennreal}) \leq \text{of-nat } x \longleftrightarrow 1 \leq x$ 
  by ( $\text{metis of-nat-le-iff of-nat-1}$ )

locale  $\text{bounded-countable-bipartite-web} = \text{countable-bipartite-web } \Gamma$ 
  for  $\Gamma :: ('v, 'more) \text{web-scheme}$  (structure)
  +
  assumes  $\text{bounded-B}: x \in A \ \Gamma \implies (\sum^+ y \in \mathbf{E} \ \{x\}. \text{weight } \Gamma \ y) < \top$ 
begin

theorem  $\text{unhindered-linkable-bounded}$ :
  assumes  $\neg \text{hindered } \Gamma$ 
  shows  $\text{linkable } \Gamma$ 
proof( $\text{cases } A \ \Gamma = \{\}$ )
  case  $\text{True}$ 
  hence  $\text{linkage } \Gamma \ (\lambda-. \ 0)$  by( $\text{auto } \text{simp } \text{add}: \text{linkage.simps}$ )
  moreover have  $\text{web-flow } \Gamma \ (\lambda-. \ 0)$  by( $\text{auto } \text{simp } \text{add}: \text{web-flow.simps}$ )
  ultimately show  $?thesis$  by  $\text{blast}$ 
next
  case  $\text{nonempty}$ :  $\text{False}$ 
  define  $A\text{-}n :: \text{nat} \Rightarrow 'v \text{ set}$  where  $A\text{-}n \ n = \text{from-nat-into } (A \ \Gamma) \ \{..n\}$  for  $n$ 
  have  $\text{fin-}A\text{-}n \ [simp]: \text{finite } (A\text{-}n \ n)$  for  $n$  by( $\text{simp } \text{add}: A\text{-}n\text{-def}$ )
  have  $A\text{-}n\text{-A}: A\text{-}n \ n \subseteq A \ \Gamma$  for  $n$  by( $\text{auto } \text{simp } \text{add}: A\text{-}n\text{-def } \text{from-nat-into}[OF$ 

```

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 “  $X \wedge (\sum^+ x \in X. \text{weight } \Gamma x) \leq$ 
 $(\sum^+ y \in Y2$   $n$   $X. \text{weight } \Gamma y) \wedge (\sum^+ y \in Y2$   $n$   $X. \text{weight } \Gamma y) \neq \top$ 
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-reft X])

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

```

```

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

  have  $s\text{-top}\ [simp]: s\ n \neq \top$  for  $n$ 
  proof -
    have  $\llbracket x \in Y2\ n\ X; X \subseteq A-n\ n \rrbracket \implies Suc\ 0 \leq card\ \{X. X \subseteq A-n\ n \wedge x \in Y2\ n\ X\}$  for  $x\ X$ 
    by(subst card-le-Suc-iff)(auto intro!: exI[where x=X] exI[where x={X. X \subseteq A-n\ n \wedge x \in Y2\ n\ X} - {X}])
    then have  $(\sum^+_{y \in Y\ n. \text{weight } \Gamma y}) \leq (\sum^+_{y \in Y\ n. \sum_{X \in Pow\ (A-n\ n). \text{weight } \Gamma y * indicator\ (Y2\ n\ X)\ y})$ 
    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  $\dots = (\sum_{X \in Pow\ (A-n\ n). \sum^+_{y \in Y\ n. \text{weight } \Gamma y * indicator\ (Y2\ n\ X)\ y})$ 
    by(subst nn-integral-sum) auto
    also have  $\dots = (\sum_{X \in Pow\ (A-n\ n). \sum^+_{y \in Y2\ n\ X. \text{weight } \Gamma y})$ 
    by(auto intro!: sum.cong nn-integral-cong simp add: nn-integral-count-space-indicator Y-def split: split-indicator)
    also have  $\dots < \top$  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  $f\ n\ xo = (case\ xo\ of\ Some\ x \Rightarrow if\ x \in A-n\ n\ then\ enn2real\ (\text{weight } \Gamma x)$ 
  else 0
    | None  $\Rightarrow enn2real\ (s\ n - sum\ (\text{weight } \Gamma)\ (A-n\ n))$ ) for  $n\ xo$ 
  define  $g :: nat \Rightarrow 'v \Rightarrow real$ 
  where  $g\ n\ y = enn2real\ (\text{weight } \Gamma y * indicator\ (Y\ n)\ y)$  for  $n\ y$ 
  define  $R :: nat \Rightarrow ('v\ option \times 'v)\ set$ 
  where  $R\ n = map\text{-prod}\ Some\ id\ '(\mathbf{E} \cap A-n\ n \times Y\ n) \cup \{None\} \times Y\ n$  for  $n$ 

```

```

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

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

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

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

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

```

from * **have** $(\sum^+ x \in X. f \ n \ x) = (\sum^+ x \in \text{Some } (the \ 'X). f \ n \ x)$ **by** *simp*
also have $\dots = (\sum^+ x \in the \ 'X. f \ n \ (\text{Some } x))$ **by** *(rule nn-integral-count-space-reindex)*
simp
also have $\dots = (\sum^+ x \in the \ 'X. weight \ \Gamma \ x)$ **using** *that False*
by *(auto 4 3 intro!: nn-integral-cong simp add: f-simps A-n'-def ennreal-enn2real-if)*
also have $\dots \leq (\sum^+ y \in Y \ 2 \ n \ (the \ 'X). weight \ \Gamma \ y)$ **using** *False that*
by *(intro le)(auto simp add: A-n'-def)*
also have $\dots \leq (\sum^+ y \in R \ n \ \text{"} X. weight \ \Gamma \ y)$ **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 $\dots = (\sum^+ y \in R \ n \ \text{"} X. g \ n \ y)$
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** $X \subseteq A-n \ n$ **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 $R \ n \subseteq A-n' \ n \times Y \ n$ **by** *(auto simp add: R-def A-n'-def)*
ultimately obtain h' **where** $\bigwedge x \ y. 0 < h' \ x \ y \implies (x, y) \in R \ n \ \bigwedge x \ y. h' \ x \ y$
 $\neq \top$
 $\bigwedge x. x \in A-n' \ n \implies (\sum^+ y \in Y \ n. h' \ x \ y) = (f \ n \ x) \ \bigwedge y. y \in Y \ n \implies (\sum^+$
 $x \in A-n' \ n. h' \ x \ y) = g \ n \ y$
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** $dom-h': \bigwedge x \ y. (x, y) \notin R \ n \implies h' \ n \ x \ y = 0$
and $fin-h' [simp]: \bigwedge x \ y. h' \ n \ x \ y \neq \top$
and $h'-f: \bigwedge x. x \in A-n' \ n \implies (\sum^+ y \in Y \ n. h' \ n \ x \ y) = f \ n \ x$
and $h'-g: \bigwedge y. y \in Y \ n \implies (\sum^+ x \in A-n' \ n. h' \ n \ x \ y) = g \ n \ y$
for n **by** *blast*

define $h :: nat \Rightarrow 'v \times 'v \Rightarrow real$
where $h \ n = (\lambda(x, y). \text{if } x \in A-n \ n \wedge y \in Y \ n \text{ then } enn2real \ (h' \ n \ (\text{Some } x)$
 $y) \text{ else } 0)$ **for** n
have $h\text{-nonneg}: 0 \leq h \ n \ x \ y$ **for** $n \ x \ y$ **by** *(simp add: h-def split-def)*
have $h\text{-notB}: h \ n \ (x, y) = 0$ **if** $y \notin B \ \Gamma$ **for** $n \ x \ y$ **using** *Y-B[of n] that* **by** *(auto*
simp add: h-def)
have $h\text{-le-weight2}: h \ n \ (x, y) \leq weight \ \Gamma \ y$ **for** $n \ x \ y$
proof *(cases x \in A-n \ n \wedge y \in Y \ n)*
case *True*
have $h' \ n \ (\text{Some } x) \ y \leq (\sum^+ x \in A-n' \ n. h' \ n \ x \ y)$
by *(rule nn-integral-ge-point)(auto simp add: A-n'-def True)*
also have $\dots \leq weight \ \Gamma \ y$ **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 (λn. h n xy)) if xy ∈ ?Q for xy unfolding bounded-def
dist-real-def
proof(rule exI strip|erule imageE|hypssubst)+
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 (λn. 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 ∈ ?Q*)(*auto simp add: conver-*
gent-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 $H': (\lambda n. \text{ennreal}\ (h\ (k\ n)\ xy)) \longrightarrow H\ xy$ **for** xy
using *H by(rule tendsto-ennrealI)*
have $H\text{-def}'$: $H\ xy = \lim (\lambda n. \text{ennreal}\ (h\ (k\ n)\ xy))$ **for** xy **by** (*metis H' limI*)

have *H-OUT*: $d\text{-OUT}\ H\ x = \text{weight}\ \Gamma\ x$ **if** $x: x \in A\ \Gamma$ **for** x
proof –
let $?w = \lambda y. \text{if } (x, y) \in \mathbf{E} \text{ then } \text{weight}\ \Gamma\ y \text{ else } 0$
have $\text{sum-w}: (\sum^+ y. \text{if } \text{edge}\ \Gamma\ x\ y \text{ then } \text{weight}\ \Gamma\ y \text{ else } 0) = (\sum^+ y \in \mathbf{E} \{x\}. \text{weight}\ \Gamma\ y)$
by(*simp add: nn-integral-count-space-indicator indicator-def of-bool-def if-distrib*
cong: if-cong)
have $(\lambda n. d\text{-OUT}\ (h\ (k\ n))\ x) \longrightarrow d\text{-OUT}\ H\ x$ **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** $n\text{-x}$ **where** $n\text{-x} = \text{to-nat-on}\ (A\ \Gamma)\ x$
have $x': x \in A\text{-}n\ (k\ n)$ **if** $n \geq n\text{-x}$ **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 $(\lambda n. d\text{-OUT}\ (h\ (k\ n))\ x) \longrightarrow \text{weight}\ \Gamma\ x$
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* $\Gamma\ H\ ..$
moreover
have *web-flow* $\Gamma\ H$ **unfolding** *web-flow-iff*
proof
show $d\text{-OUT}\ H\ x \leq \text{weight}\ \Gamma\ x$ **for** x
by(*cases x ∈ A Γ*)(*simp-all add: H-OUT[unfolded d-OUT-def] H-outside*
d-OUT-def)

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

```

      unfolding d-IN-def by(rule nn-integral-liminf) simp
    also have ... ≤ liminf (λn. 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 ∉ 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 (λn. ennreal (h (k n) e)) ⟶ 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 ∈ V - B Γ ⟹ (∑+ y ∈ E “ {x}. weight Γ y) < ⊤
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 ∈ A ?Γ
    then obtain x' where x: x = Inl x' and x': vertex Γ x' x' ∉ B Γ by auto
    have E_Γ “ {x} ⊆ Inr “ ({x'} ∪ (E “ {x'})) using x
      by(auto simp add: bipartite-web-of-def vertex-def split: sum.split-asm)
    hence (∑+ y ∈ E_Γ “ {x}. weight ?Γ y) ≤ (∑+ y ∈ ... weight ?Γ y)
      by(auto simp add: nn-integral-count-space-indicator intro!: nn-integral-mono
split: split-indicator)
    also have ... = (∑+ y ∈ {x'} ∪ (E “ {x'}). weight (bipartite-web-of Γ) (Inr
y))
      by(rule nn-integral-count-space-reindex)(auto)
    also have ... ≤ weight Γ x' + (∑+ y ∈ E “ {x'}. weight Γ y)

```

```

    apply(subst (1 2) nn-integral-count-space-indicator, simp, simp)
    apply(cases  $\neg$  edge  $\Gamma$   $x'$   $x'$ )
    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  $(\sum^+ y \in \mathbf{E}_{\mathcal{T}} \text{ ``}\{x\}. \text{ weight } ?\Gamma y) < \top$  .
qed
qed

```

theorem *loose-linkable-bounded:*

```

  assumes loose  $\Gamma$ 
  shows linkable  $\Gamma$ 
proof -
  interpret bi: bounded-countable-bipartite-web bipartite-web-of  $\Gamma$  by(rule bounded-countable-bipartite-web-of)
  have  $\neg$  hindered (bipartite-web-of  $\Gamma$ ) using assms by(rule unhindered-bipartite-web-of)
  then have linkable (bipartite-web-of  $\Gamma$ )
    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  $x \in \mathbf{V}_{\text{quotient-web } \Gamma f} - B$  (quotient-web  $\Gamma$   $f$ )
    then have  $x: x \in \mathbf{V} - B \Gamma$  by(auto dest: vertex-quotient-webD)
    have  $(\sum^+ y \in \mathbf{E}_{\mathcal{T}} \text{ ``}\{x\}. \text{ weight } ?\Gamma y) \leq (\sum^+ y \in \mathbf{E} \text{ ``}\{x\}. \text{ weight } \Gamma y)$ 
      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  $(\sum^+ y \in \mathbf{E}_{\mathcal{T}} \text{ ``}\{x\}. \text{ weight } ?\Gamma y) < \top$  .
  qed
qed

```

lemma *ex-orthogonal-current:*

```

 $\exists f S. \text{ web-flow } \Gamma f \wedge \text{ separating } \Gamma S \wedge \text{ orthogonal-current } \Gamma f S$ 
proof -
  from ex-maximal-wave[OF countable]
  obtain  $f$  where  $f$ : current  $\Gamma f$ 
  and  $w$ : wave  $\Gamma f$ 
  and maximal:  $\bigwedge w. [\text{ current } \Gamma w; \text{ wave } \Gamma w; f \leq w] \implies f = w$  by blast
  from ex-trimming[OF  $f$   $w$  countable weight-finite] obtain  $h$  where  $h$ : trimming
 $\Gamma f$   $h$  ..

```

```

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

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

end

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

context antiparallel-edges begin

lemma  $\Delta''$ -bounded-countable-network: bounded-countable-network  $\Delta''$ 
if  $\bigwedge x. \llbracket x \in \mathbf{V}; x \neq \text{source } \Delta; x \neq \text{sink } \Delta \rrbracket \implies d\text{-OUT } (\text{capacity } \Delta) \ x < \top$ 
proof –
interpret  $ae$ : countable-network  $\Delta''$  by(rule  $\Delta''$ -countable-network)
show ?thesis
proof
fix  $x$ 
assume  $x$ :  $x \in \mathbf{V}_{\Delta''}$  and not-source:  $x \neq \text{source } \Delta''$  and not-sink:  $x \neq \text{sink } \Delta''$ 
from  $x$  consider (Vertex)  $x'$  where  $x = \text{Vertex } x' \ x' \in \mathbf{V} \mid (\text{Edge}) \ y \ z$  where
 $x = \text{Edge } y \ z \ \text{edge } \Delta \ y \ z$ 
unfolding  $\mathbf{V}$ - $\Delta''$  by auto
then show  $d\text{-OUT } (\text{capacity } \Delta'') \ x < \top$ 
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*: $\bigwedge x. \neg \text{edge } \Delta x \text{ (source } \Delta)$

and *sink-out*: $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$

and *undead*: $\bigwedge x y. \text{edge } \Delta x y \implies (\exists z. \text{edge } \Delta y z) \vee (\exists z. \text{edge } \Delta z x)$

and *source-sink*: $\neg \text{edge } \Delta (\text{source } \Delta) (\text{sink } \Delta)$

and *no-loop*: $\bigwedge x. \neg \text{edge } \Delta x x$

shows *bounded-countable-web* (*web-of-network* Δ) (**is** *bounded-countable-web* $?T$)

proof –

interpret *web*: *countable-web* $?T$ **by**(*rule countable-web-web-of-network*) *fact+*

show *?thesis*

proof

fix *e*

assume $e \in V_{?T} - B \text{ } ?T$

then obtain $x y$ **where** $e: e = (x, y)$ **and** $xy: \text{edge } \Delta x y$ **by**(*cases e*) *simp*

from xy **have** $y: y \neq \text{source } \Delta$ **using** *source-in[of x]* **by** *auto*

have *out-sink*: $d\text{-OUT } (\text{capacity } \Delta) (\text{sink } \Delta) = 0$ **unfolding** *d-OUT-def*

by(*auto simp add: nn-integral-0-iff-AE AE-count-space sink-out intro!: capacity-outside*)

have $\mathbf{E}_{?T} \text{ “ } \{e\} \subseteq \mathbf{E} \cap \{y\} \times UNIV$ **using** *e* **by** *auto*

hence $(\sum^+ e' \in \mathbf{E}_{?T} \text{ “ } \{e\}. \text{weight } ?T e') \leq (\sum^+ e \in \mathbf{E} \cap \{y\} \times UNIV. \text{capacity } \Delta e)$ **using** *e*

by(*auto simp add: nn-integral-count-space-indicator intro!: nn-integral-mono split: split-indicator*)

also have $\dots \leq (\sum^+ e \in \text{Pair } y \text{ ‘ } UNIV. \text{capacity } \Delta e)$

by(*auto simp add: nn-integral-count-space-indicator intro!: nn-integral-mono split: split-indicator*)

also have $\dots = d\text{-OUT } (\text{capacity } \Delta) y$ **unfolding** *d-OUT-def*

by(*rule nn-integral-count-space-reindex*) *simp*

also have $\dots < \top$ **using** *out[of y]* $xy y$ *out-sink* **by**(*cases y = sink Δ*)(*auto simp add: vertex-def*)

finally show $(\sum^+ e' \in \mathbf{E}_{?T} \text{ “ } \{e\}. \text{weight } ?T e') < \top$.

qed

qed

context **begin**

qualified lemma *max-flow-min-cut'-bounded*:

assumes *source-in*: $\bigwedge x. \neg \text{edge } \Delta x \text{ (source } \Delta)$

and *sink-out*: $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$

and *undead*: $\bigwedge x y. \text{edge } \Delta x y \implies (\exists z. \text{edge } \Delta y z) \vee (\exists z. \text{edge } \Delta z x)$

and *source-sink*: $\neg \text{edge } \Delta (\text{source } \Delta) (\text{sink } \Delta)$

and *no-loop*: $\bigwedge x. \neg \text{edge } \Delta x x$

and *capacity-pos*: $\bigwedge e. e \in \mathbf{E} \implies \text{capacity } \Delta e > 0$

shows $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$

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*: $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$
and *source-in*: $\bigwedge x. \neg \text{edge } \Delta x (\text{source } \Delta)$
and *no-loop*: $\bigwedge x. \neg \text{edge } \Delta x x$
and *capacity-pos*: $\bigwedge e. e \in \mathbf{E} \implies \text{capacity } \Delta e > 0$
shows $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$

proof –

interpret *antiparallel-edges* $\Delta \dots$

interpret Δ'' : *bounded-countable-network* Δ'' **by**(*rule* Δ'' -*bounded-countable-network*)(*rule out*)

have $\exists f S. \text{flow } \Delta'' f \wedge \text{cut } \Delta'' S \wedge \text{orthogonal } \Delta'' f S$

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* $\Delta'' f$ **and** cut : *cut* $\Delta'' S$ **and** *ortho*: *orthogonal* $\Delta'' f S$ **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*: $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$
and *source-in*: $\bigwedge x. \neg \text{edge } \Delta x (\text{source } \Delta)$
and *capacity-pos*: $\bigwedge e. e \in \mathbf{E} \implies \text{capacity } \Delta e > 0$
shows $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$

proof –

interpret *antiparallel-edges* $\Delta \dots$

interpret Δ'' : *bounded-countable-network* Δ'' **by**(*rule* Δ'' -*bounded-countable-network*)(*rule out*)

have $\exists f S. \text{flow } \Delta'' f \wedge \text{cut } \Delta'' S \wedge \text{orthogonal } \Delta'' f S$

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* $\Delta'' f$ **and** cut : *cut* $\Delta'' S$ **and** *ortho*: *orthogonal* $\Delta'' f S$ **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  $x: x \in V_{\Delta'''} \text{ and not-source: } x \neq \text{source } \Delta''' \text{ and not-sink: } x \neq \text{sink } \Delta'''$ 
    from  $x$  have  $x': x \in V$  by (auto simp add: vertex-def)
    have  $d\text{-OUT (capacity } \Delta''') x \leq d\text{-OUT (capacity } \Delta) x$  by (rule d-OUT-mono)
simp
    also have  $\dots < \top$  using  $x' \text{ not-source not-sink}$  by (simp add: out)
    finally show  $d\text{-OUT (capacity } \Delta''') x < \top$  .
qed
qed

theorem max-flow-min-cut-bounded:
   $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$ 
proof -
  interpret  $\Delta'$ : bounded-countable-network  $\Delta'''$  by (rule  $\Delta'''$ -bounded-countable-network)
  have  $\exists f S. \text{flow } \Delta''' f \wedge \text{cut } \Delta''' S \wedge \text{orthogonal } \Delta''' f S$  by (rule  $\Delta'.\text{max-flow-min-cut}'''$ -bounded)
auto
  then obtain  $f S$  where  $f: \text{flow } \Delta''' f$  and  $\text{cut: cut } \Delta''' S$  and  $\text{ortho: orthogonal } \Delta''' f S$  by blast
  from  $\text{flow-}\Delta'''$  [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 $\text{cleanup} :: 'a \text{ flow} \Rightarrow 'a \text{ flow}$
where $\text{cleanup } f = (\lambda(a, b). \text{ if } f(a, b) > f(b, a) \text{ then } f(a, b) - f(b, a) \text{ else } 0)$

lemma cleanup-simps [simp]:
 $\text{cleanup } f(a, b) = (\text{ if } f(a, b) > f(b, a) \text{ then } f(a, b) - f(b, a) \text{ else } 0)$
 by (simp add: cleanup-def)

lemma $\text{value-flow-cleanup}$:
 assumes [simp]: $\bigwedge x. f(x, \text{source } \Delta) = 0$
 shows $\text{value-flow } \Delta (\text{cleanup } f) = \text{value-flow } \Delta f$
 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  $\langle d-OUT\ f\ x \neq \top \rangle$ 
    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 ... =  $(\sum^+ y \in ?M. f(x, y)) - (\sum^+ y \in ?M. f(y, x))$  using finite-IN
    by(subst nn-integral-diff)(auto simp add: AE-count-space)
  also have ... =  $(d-OUT\ f\ x - (\sum^+ y \in \{y. f(x, y) \leq f(y, x)\}. f(x, y))) -$ 
 $(\sum^+ y \in ?M. f(y, x))$ 
    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 ... =  $(d-IN\ f\ x - (\sum^+ y \in ?M. f(y, x))) - (\sum^+ y \in \{y. f(x, y) \leq f$ 
 $(y, x)\}. f(x, y))$ 
    using KIR by(simp add: diff-diff-commute-ennreal)
  also have ... =  $(\sum^+ y \in \{y. f(x, y) \leq f(y, x)\}. f(y, x)) - (\sum^+ y \in \{y. f(x,$ 
 $y) \leq f(y, x)\}. f(x, y))$ 
    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 ... =  $(\sum^+ y \in \{y. f(x, y) \leq f(y, x)\}. f(y, x) - f(x, y))$  using
finite-OUT
    by(subst nn-integral-diff)(auto simp add: AE-count-space)
  also have ... = 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  $\Delta$ 
  for  $\Delta :: ('v, 'more)$  network-scheme (structure)
  +
  assumes finite-capacity:  $\bigwedge x. x \neq \text{sink } \Delta \implies d\text{-IN } (\text{capacity } \Delta) x \neq \top \vee d\text{-OUT } (\text{capacity } \Delta) x \neq \top$ 
  and no-loop:  $\bigwedge x. \neg \text{edge } \Delta x x$ 
  and source-in:  $\bigwedge x. \neg \text{edge } \Delta x (\text{source } \Delta)$ 
begin

lemma source-in-not-cycle:
  assumes cycle  $\Delta p$ 
  shows  $(x, \text{source } \Delta) \notin \text{set } (\text{cycle-edges } p)$ 
using cycle-edges-edges[OF assms] source-in[of x] by(auto)

lemma source-out-not-cycle:
   $\text{cycle } \Delta p \implies (\text{source } \Delta, x) \notin \text{set } (\text{cycle-edges } p)$ 
by(auto dest: cycle-leave-ex-enter source-in-not-cycle)

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

lemma flowD-finite-IN:
  assumes  $f: \text{flow } \Delta f$  and  $x: x \neq \text{sink } \Delta$ 
  shows  $d\text{-IN } f x \neq \text{top}$ 
proof(cases  $x = \text{source } \Delta$ )
  case True thus ?thesis by(simp add: flowD-source-IN[OF f])
next
  case False
  from finite-capacity[OF x] show ?thesis
  proof
    assume *:  $d\text{-IN } (\text{capacity } \Delta) x \neq \top$ 
    from flowD-capacity[OF f] have  $d\text{-IN } f x \leq d\text{-IN } (\text{capacity } \Delta) x$  by(rule d-IN-mono)
    also have  $\dots < \top$  using * by (simp add: less-top)
    finally show ?thesis by simp
  next
    assume *:  $d\text{-OUT } (\text{capacity } \Delta) x \neq \top$ 
    have  $d\text{-IN } f x = d\text{-OUT } f x$  using flowD-KIR[OF f False x] by simp
    also have  $\dots \leq d\text{-OUT } (\text{capacity } \Delta) x$  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  $\Delta$   $f$   $x \neq \text{source } \Delta$   $x \neq \text{sink } \Delta$ 
  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  $g :: 'v \text{ flow}$ 
  assumes  $g: \text{flow } \Delta$   $g$ 
  and  $g\text{-finite: value-flow } \Delta$   $g \neq \top$ 
  and  $\text{nontrivial: } \mathbf{V} - \{\text{source } \Delta, \text{sink } \Delta\} \neq \{\}$ 
begin

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

lemma  $g\text{-loop [simp]: } g\ (x, x) = 0$ 
by (rule  $g\text{-outside}$ ) (simp add: no-loop)

lemma  $\text{finite-IN-}g: x \neq \text{sink } \Delta \implies d\text{-IN } g\ x \neq \text{top}$ 
by (rule flowD-finite-IN[OF  $g$ ])

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

lemma  $g\text{-source-in [simp]: } g\ (x, \text{source } \Delta) = 0$ 
by (rule  $g\text{-outside}$ ) (simp add: source-in)

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

definition  $\text{enum-}v :: \text{nat} \Rightarrow 'v$ 

```

```

where enum-v n = from-nat-into ( $\mathbf{V} - \{\text{source } \Delta, \text{sink } \Delta\}$ ) (fst (prod-decode n))

lemma range-enum-v: range enum-v  $\subseteq \mathbf{V} - \{\text{source } \Delta, \text{sink } \Delta\}$ 
using from-nat-into[OF nontrivial] by(auto simp add: enum-v-def)

lemma enum-v-repeat:
  assumes x:  $x \in \mathbf{V} \ x \neq \text{source } \Delta \ x \neq \text{sink } \Delta$ 
  shows  $\exists i' > i. \text{enum-v } i' = x$ 
proof -
  let ?V =  $\mathbf{V} - \{\text{source } \Delta, \text{sink } \Delta\}$ 
  let ?n = to-nat-on ?V x
  let ?A =  $\{?n\} \times (\text{UNIV} :: \text{nat set})$ 
  from x have x':  $x \in \mathbf{V} - \{\text{source } \Delta, \text{sink } \Delta\}$  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  $i' > i \ i' \in \text{prod-encode } ' \ ?A$ 
  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  $\langle i' > i \rangle$  show ?thesis by blast
qed

fun h-plus :: nat  $\Rightarrow$  'v edge  $\Rightarrow$  ennreal
where
  h-plus 0 (x, y) = (if  $x = \text{source } \Delta$  then g (x, y) else 0)
| h-plus (Suc i) (x, y) =
  (if enum-v (Suc i) = x  $\wedge$  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  $\leq$  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:  $x = \text{enum-v } (\text{Suc } i)$ 
    assume less: d-OUT (h-plus i) x < d-IN (h-plus i) x
    from enum have x:  $x \neq \text{source } \Delta \ x \neq \text{sink } \Delta$  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 $d\text{-OUT } (h\text{-plus } i) \ x \leq d\text{-OUT } g \ x$ **by** (rule $d\text{-OUT-mono}$) (rule Suc.IH)
also have $\dots < \text{top}$ **using** $\text{finite-OUT-g[of } x]$ x **by** (simp add: less-top)
finally have $d\text{-OUT } (h\text{-plus } i) \ x \neq \top$ **by** simp
then have $\text{shares-eq: shares} = (\sum^+ y. g \ (x, y) - h\text{-plus } i \ (x, y))$ **unfolding**
 $\text{shares-def } d\text{-OUT-def}$
by (subst nn-integral-diff) (simp-all add: $\text{AE-count-space } \text{Suc.IH}$)

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

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

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

lemma $h\text{-plus-not-infty [simp]: } h\text{-plus } i \ e \neq \text{top}$
using $h\text{-plus-le-g[of } i \ e]$ **by** (auto simp: top-unique)

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

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

lemma $h\text{-plus-mono}' : i \leq j \implies h\text{-plus } i \ e \leq h\text{-plus } j \ e$
by(induction rule: dec-induct)(auto intro: h-plus-mono order-trans)

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

```

1 = d-IN (h-plus i) x using x
  by (simp add: Suc)
  finally show ?thesis by simp
qed
also have ... ≤ 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 ≠ 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 ≠ source Δ and sink: x ≠ sink Δ using range-enum-v
by auto
    let ?r = λy. (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 + (∑+ 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 ≠ source Δ x ≠ 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 (∑+ y. ?r y) =
      (d-IN (h-plus i) x - d-OUT (h-plus i) x) * ((∑+ 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 (∑+ 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:  $(\sum^+ e. \text{h-plus } i \ e) \neq \text{top}$ 
proof(induction i)
  case 0
    have  $(\sum^+ e \in \text{UNIV}. \text{h-plus } 0 \ e) = (\sum^+ (x, y). \text{h-plus } 0 \ (x, y))$ 
    by(simp del: h-plus.simps)
    also have  $\dots = (\sum^+ (x, y) \in \text{range } (\text{Pair } (\text{source } \Delta)). \text{h-plus } 0 \ (x, y))$ 
    by(auto simp add: nn-integral-count-space-indicator intro!: nn-integral-cong)
    also have  $\dots = \text{value-flow } \Delta \ g$  by(simp add: d-OUT-def nn-integral-count-space-reindex)
    also have  $\dots < \top$  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  $\neq \text{source } \Delta$  xi  $\neq \text{sink } \Delta$  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  $(\sum^+ e \in \text{UNIV}. \text{h-plus } (\text{Suc } i) \ e) = (\sum^+ e. \text{h-plus } i \ e)$ 
        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  $(\sum^+ e. \text{h-plus } (\text{Suc } i) \ e) =$ 
           $(\sum^+ e \in \text{UNIV}. \text{h-plus } i \ e) + (\sum^+ (x, y). ((g \ (x, y) - \text{h-plus } i \ (x, y)) * (d-IN \ (h-plus \ i) \ x - d-OUT \ (h-plus \ i) \ x) / (d-OUT \ g \ x - d-OUT \ (h-plus \ i) \ x))$ 
           $* \text{indicator } (\text{range } (\text{Pair } xi)) \ (x, y))$ 
          (is - = ?IH + ?rest is - = - +  $\int^+ (x, y). ?f \ x \ y \ * \ - \partial-$ ) 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 =  $(\sum^+ (x, y) \in \text{range } (\text{Pair } xi). ?f \ x \ y)$ 
          by(simp add: nn-integral-count-space-indicator split-def)
          also have  $\dots = (\sum^+ y. ?f \ xi \ y)$  by(simp add: nn-integral-count-space-reindex)
          also have  $\dots = (\sum^+ y. g \ (xi, y) - \text{h-plus } i \ (xi, y)) * ((d-IN \ (h-plus \ i) \ xi - d-OUT \ (h-plus \ i) \ xi) / (d-OUT \ g \ xi - d-OUT \ (h-plus \ i) \ xi))$ 
          (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  $\dots * ?factor = (d-IN \ (h-plus \ i) \ xi - d-OUT \ (h-plus \ i) \ xi)$  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, sym-
metric] 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  $\Delta$ )

```

```

lemma cycle-enum-cycle [simp]: cycles  $\Delta \neq \{\}$   $\implies$  cycle  $\Delta$  (enum-cycle n)
unfolding enum-cycle-def using from-nat-into[of cycles  $\Delta$  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 [[simp proc 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 ≠ 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 ≤ h-minus-aux (Suc j) e
proof(cases e ∈ set (cycle-edges (enum-cycle j)) = True)
  case True
  have h-minus-aux j e + 0 ≤ 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 Δ ≠ {}
  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' ∈ set (cycle-edges C)}

  have d-OUT (h-minus-aux (Suc j)) x =
    (∑+ y. h-minus-aux j (x, y) + (if (x, y) ∈ 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 + (∑+ 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 = (∑+ e∈range (Pair x). δ * indicator {(x', y). (x', y) ∈ set
  (cycle-edges C) ∧ 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  $\delta * \dots = (\sum^+ e \in \text{range } (\lambda x'. (x', x)). \delta * \text{indicator } \{(x', y). (x', y) \in \text{set } (\text{cycle-edges } C) \wedge y = x\} e)$ 
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  $\dots = (\sum^+ x'. \delta * \text{indicator } (\text{set } (\text{cycle-edges } C)) (x', x))$ 
by(auto simp add: nn-integral-count-space-reindex intro!: nn-integral-cong split:
split-indicator)
also have  $d\text{-OUT } (h\text{-minus-aux } j) x + \dots = (\sum^+ x'. h\text{-minus-aux } j (x', x) + \delta * \text{indicator } (\text{set } (\text{cycle-edges } C)) (x', x))$ 
unfolding Suc.IH d-IN-def by(simp add: nn-integral-add[symmetric])
also have  $\dots = d\text{-IN } (h\text{-minus-aux } (Suc j)) x$  unfolding d-IN-def
by(auto intro!: nn-integral-cong simp add:  $\delta$ -def C-def split: split-indicator)
finally show ?case .
qed

```

```

lemma h-minus-aux-source:
assumes cycles  $\Delta \neq \{\}$ 
shows  $h\text{-minus-aux } j (\text{source } \Delta, y) = 0$ 
proof(induction j)
case 0 thus ?case by simp
next
case (Suc j)
have  $(\text{source } \Delta, y) \notin \text{set } (\text{cycle-edges } (\text{enum-cycle } j))$ 
proof
assume *:  $(\text{source } \Delta, y) \in \text{set } (\text{cycle-edges } (\text{enum-cycle } j))$ 
have cycle: cycle  $\Delta$  (enum-cycle j) using assms by(rule cycle-enum-cycle)
from cycle-leave-ex-enter[OF this *]
obtain z where  $(z, \text{source } \Delta) \in \text{set } (\text{cycle-edges } (\text{enum-cycle } j))$  ..
with cycle-edges-edges[OF cycle] have  $(z, \text{source } \Delta) \in \mathbf{E}$  ..
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  $C \equiv \text{enum-cycle } j$ 
assumes cycles  $\Delta \neq \{\}$ 
shows  $\exists e \in \text{set } (\text{cycle-edges } C). h\text{-minus-aux } (Suc j) e = h' e$ 
proof -
let ?A =  $\{h' e' - h\text{-minus-aux } j e' \mid e' \in \text{set } (\text{cycle-edges } C)\}$ 
from assms have cycle  $\Delta$  C by auto
from cycle-edges-not-Nil[OF this] have Min ?A  $\in$  ?A using [[simproc add: fi-
nite-Collect]]
by(intro Min-in)(fastforce simp add: neq-Nil-conv)+
then obtain e' where  $e' \in \text{set } (\text{cycle-edges } C)$ 

```

```

    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  $\Delta \neq \{\}$ 
shows d-OUT (h-minus i) x = d-IN (h-minus i) x
proof(induction i)
  case (Suc i)
    let ?h' =  $\lambda e. h\text{-plus } (Suc\ i)\ e - h\text{-minus } i\ e$ 
    have d-OUT ( $\lambda e. h\text{-minus } (Suc\ i)\ e$ ) x = d-OUT (h-minus i) x + d-OUT ( $\lambda e. SUP\ j. h\text{-minus-}aux\ ?h'\ j\ e$ ) x
    by(simp add: d-OUT-add SUP-upper2)
    also have d-OUT ( $\lambda e. SUP\ j. h\text{-minus-}aux\ ?h'\ j\ e$ ) 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  $\dots = (SUP\ j. d\text{-IN } (h\text{-minus-}aux\ ?h'\ j)\ x)$ 
    by(rule SUP-cong[OF refl](rule d-OUT-h-minus-}aux[OF finite-h' cycles]))
    also have  $\dots = d\text{-IN } (\lambda e. SUP\ j. h\text{-minus-}aux\ ?h'\ j\ e)\ x$ 
    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 +  $\dots = d\text{-IN } (\lambda e. h\text{-minus } (Suc\ i)\ e)\ x$  using
Suc.IH
    by(simp add: d-IN-add SUP-upper2)
    finally show ?case .
qed simp

```

```

lemma h-minus-source:
  assumes cycles  $\Delta \neq \{\}$ 
  shows h-minus n (source  $\Delta$ , 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  $\Delta$ ) = 0
using h-minus-le-h-plus[of i (x, source  $\Delta$ )] by simp

```

```

lemma h-minus-OUT-finite [simp]: d-OUT (h-minus i) x  $\neq top$ 
proof –
  have d-OUT (h-minus i) x  $\leq d\text{-OUT } (h\text{-plus } i)\ x$  by(rule d-OUT-mono)(rule h-minus-le-h-plus)
  also have  $\dots < \top$  by (simp add: less-top[symmetric])
  finally show ?thesis by simp
qed

```

```

lemma h-minus-cycle:
  assumes cycle  $\Delta\ C$ 
  shows  $\exists e \in set\ (cycle\text{-edges } C). h\text{-minus } i\ e = h\text{-plus } i\ e$ 
proof(cases i)
  case (Suc i)
    let ?h' =  $\lambda e. h\text{-plus } (Suc\ i)\ e - h\text{-minus } i\ e$ 
    from assms have cycles: cycles  $\Delta \neq \{\}$  by auto
    with assms from-nat-into-surj[of cycles  $\Delta\ C$ ] obtain j where j: C = enum-cycle
    by(auto simp add: enum-cycle-def)

```

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

abbreviation $\text{lim-h-plus} :: 'v \text{ edge} \Rightarrow \text{ennreal}$
where $\text{lim-h-plus } e \equiv \text{SUP } n. h\text{-plus } n e$

abbreviation $\text{lim-h-minus} :: 'v \text{ edge} \Rightarrow \text{ennreal}$
where $\text{lim-h-minus } e \equiv \text{SUP } n. h\text{-minus } n e$

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

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

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

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

lemma $\text{lim-h-minus-IN-finite}$ [*simp*]:
assumes $x \neq \text{sink } \Delta$
shows $d\text{-IN } \text{lim-h-minus } x \neq \text{top}$
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 $d\text{-OUT } \lim\text{-}h\text{-minus } x = d\text{-IN } \lim\text{-}h\text{-minus } x$
proof –
 have $d\text{-OUT } \lim\text{-}h\text{-minus } x = (\text{SUP } n. d\text{-OUT } (h\text{-minus } n) x)$
 by(rule $d\text{-OUT-monotone-convergence-SUP incseq-SucI le-funI h-minus-mono}$)+
 also have $\dots = (\text{SUP } n. d\text{-IN } (h\text{-minus } n) x)$ **using** *cycles* **by**(*simp add:*
d-OUT-h-minus)
 also have $\dots = d\text{-IN } \lim\text{-}h\text{-minus } x$
 by(rule $d\text{-IN-monotone-convergence-SUP[symmetric] incseq-SucI le-funI h-minus-mono}$)+
 finally **show** *?thesis* .
qed

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

lemma $h\text{-le-}\lim\text{-}h\text{-plus}$: $h \ e \leq \lim\text{-}h\text{-plus } e$
by (*simp add: h-def*)

lemma $h\text{-le-}g$: $h \ e \leq g \ e$
using $h\text{-le-}\lim\text{-}h\text{-plus}[of \ e] \ \lim\text{-}h\text{-plus-le-}g[of \ e]$ **by** *simp*

lemma $flow\text{-}h$: $flow \ \Delta \ h$

proof
 fix e
 have $h \ e \leq \lim\text{-}h\text{-plus } e$ **by**(rule $h\text{-le-}\lim\text{-}h\text{-plus}$)
 also have $\dots \leq g \ e$ **by**(rule $\lim\text{-}h\text{-plus-le-}g$)
 also have $\dots \leq \text{capacity } \Delta \ e$ **using** g **by**(rule $flowD\text{-capacity}$)
 finally **show** $h \ e \leq \dots$.
next
 fix x
 assume $x \neq \text{source } \Delta \ x \neq \text{sink } \Delta$
 then **show** $KIR \ h \ x$
 by (*cases* $\text{cycles } \Delta = \{\}$)
 (*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 $value\text{-}h\text{-plus}$: $value\text{-}flow \ \Delta \ (h\text{-plus } i) = value\text{-}flow \ \Delta \ g \ (\text{is } ?lhs = ?rhs)$

proof(rule *antisym*)
 show $?lhs \leq ?rhs$ **by**(rule $d\text{-OUT-mono}$)(rule $h\text{-plus-le-}g$)

 have $?rhs \leq value\text{-}flow \ \Delta \ (h\text{-plus } 0)$
 by(*auto simp add: d-OUT-def cong: if-cong intro!: nn-integral-mono*)
 also have $\dots \leq value\text{-}flow \ \Delta \ (h\text{-plus } i)$
 by(rule $d\text{-OUT-mono}$)(rule $h\text{-plus-mono'}$; *simp*)
 finally **show** $?rhs \leq ?lhs$.
qed

lemma $value\text{-}h$: $value\text{-}flow \ \Delta \ h = value\text{-}flow \ \Delta \ g \ (\text{is } ?lhs = ?rhs)$

proof(rule *antisym*)

have $?lhs \leq \text{value-flow } \Delta \text{ lim-h-plus}$ **using** *ennreal-minus-mono*
by(*fastforce simp add: h-def intro!: d-OUT-mono*)
also have $\dots \leq ?rhs$ **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* :: $\text{nat} \Rightarrow 'v \text{ edge} \Rightarrow \text{ennreal}$
where *h-diff* $i \ e = h\text{-plus } i \ e - (\text{if } \text{cycles } \Delta \neq \{\} \text{ then } h\text{-minus } i \ e \text{ else } 0)$

lemma *d-IN-h-source* [*simp*]: $d\text{-IN } (h\text{-diff } i) (\text{source } \Delta) = 0$
by(*simp add: d-IN-def h-diff-def cong del: if-weak-cong*)

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

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

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

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

lemma *h-diff-OUT-le-IN*:
assumes $x \neq \text{source } \Delta$
shows $d\text{-OUT } (h\text{-diff } i) \ x \leq d\text{-IN } (h\text{-diff } i) \ x$
proof(*cases cycles Δ ≠ {}*)
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 $d\text{-OUT } (h\text{-diff } i) \ x = d\text{-OUT } (h\text{-plus } i) \ x - d\text{-OUT } (h\text{-minus } i) \ x$
unfolding *h-diff-def[abs-def]* **using** *assms*
by (*simp add: h-minus-le-h-plus d-OUT-diff*)
also have $\dots \leq d\text{-IN } (h\text{-plus } i) \ x - d\text{-IN } (h\text{-minus } i) \ x$ **using** *cycles assms*
by(*intro ennreal-minus-mono h-plus-OUT-le-IN*)(*simp-all add: d-OUT-h-minus*)
also have $\dots = d\text{-IN } (h\text{-diff } i) \ x$ **using** *cycles*

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

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

lemma $d\text{-IN-h-le-value'}$: $d\text{-IN}\ (h\text{-diff}\ i)\ x \leq value\text{-flow}\ \Delta\ (h\text{-plus}\ i)$

proof –
let $?supp = support\text{-flow}\ (h\text{-diff}\ i)$
define X **where** $X = \{y. (y, x) \in ?supp^{\wedge*}\} - \{x\}$

{ 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** $rtrancl\text{-path}\ (\lambda x\ y. (x, y) \in ?supp)\ y\ p'\ x$
unfolding $rtrancl\text{-def}\ rtranclp\text{-eq-rtrancl-path}$ **by** $auto$
then obtain p **where** $p: rtrancl\text{-path}\ (\lambda x\ y. (x, y) \in ?supp)\ y\ p\ x$
and $distinct: distinct\ (y \# p)$ **by** ($rule\ rtrancl\text{-path-distinct}$)
with neq **have** $p \neq []$ **by** ($auto\ elim: rtrancl\text{-path.cases}$)

from xy **have** $(x, y) \in E$ **using** $supp\text{-h-diff-edges}[of\ i]$ **by** ($auto$)
moreover from p **have** $path\ \Delta\ y\ p\ x$
by ($rule\ rtrancl\text{-path-mono}$) ($auto\ dest: supp\text{-h-diff-edges}[THEN\ subsetD]$)
ultimately have $path\ \Delta\ x\ (y \# p)\ x$ **by** ($auto\ intro: rtrancl\text{-path.intros}$)
hence $cycle: cycle\ \Delta\ (y \# p)$ **using** $- distinct$ **by** ($rule\ cycle$) $simp$
from $h\text{-diff-cycle}[OF\ this, of\ i]$ **obtain** e
where $e: e \in set\ (cycle\text{-edges}\ (y \# p))$ **and** $0: h\text{-diff}\ i\ e = 0$ **by** $blast$
from e **obtain** n **where** $e': e = ((y \# p) ! n, (p @ [y]) ! n)$ **and** $n: n < Suc\ (length\ p)$
by ($auto\ simp\ add: cycle\text{-edges-def}\ set\text{-zip}$)
have $e \in ?supp$
proof ($cases\ n = length\ p$)
case $True$
with $rtrancl\text{-path-last}[OF\ p]\ \langle p \neq [] \rangle$ **have** $(y \# p) ! n = x$
by ($cases\ p$) ($simp\text{-all}\ add: last\text{-conv-nth}\ del: last.simps$)
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 rtranc1-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  $(x, y) \notin ?supp$  using  $x y$ 
    by(auto simp add: X-def intro: converse-rtranc1-into-rtranc1)
  then have  $h\text{-diff } i (x, y) = 0$ 
    by(simp add: support-flow.simps) }
  note acyclic = this

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

let ?diff =  $\lambda x. (\sum^+ y. h\text{-diff } i (x, y) * \text{indicator } X x * \text{indicator } X y)$ 
have finite2:  $(\sum^+ x. ?diff x) \neq \text{top}$  (is ?lhs  $\neq$  -)
proof -
  have ?lhs  $\leq (\sum^+ x \in UNIV. \sum^+ y. h\text{-plus } i (x, y))$ 
    by(intro nn-integral-mono)(auto simp add: h-diff-def split: split-indicator)
  also have  $\dots = (\sum^+ e. h\text{-plus } i e)$  by(rule nn-integral-fst-count-space)
  also have  $\dots < \top$  by(simp add: h-plus-sum-finite less-top[symmetric])
  finally show ?thesis by simp
qed
have finite1: ?diff  $x \neq \text{top}$  for  $x$ 
  using finite2 by(rule neq-top-trans)(rule nn-integral-ge-point, simp)
have finite3:  $(\sum^+ x. d\text{-OUT } (h\text{-diff } i) x * \text{indicator } (X - \{\text{source } \Delta\}) x) \neq \top$ 
(is ?lhs  $\neq$  -)
proof -
  have ?lhs  $\leq (\sum^+ x \in UNIV. \sum^+ y. h\text{-plus } i (x, y))$  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  $\dots = (\sum^+ e. h\text{-plus } i e)$  by(rule nn-integral-fst-count-space)
  also have  $\dots < \top$  by(simp add: h-plus-sum-finite less-top[symmetric])
  finally show ?thesis by simp
qed

have d-IN  $(h\text{-diff } i) x = (\sum^+ y. h\text{-diff } i (y, x) * \text{indicator } X y)$  unfolding
d-IN-def
  by(rule nn-integral-cong)(simp add: in-X split: split-indicator)
also have  $\dots \leq (\sum^+ x \in - X. \sum^+ y. h\text{-diff } i (y, x) * \text{indicator } X y)$ 
  by(rule nn-integral-ge-point)(simp add: X-def)
also have  $\dots = (\sum^+ x \in UNIV. \sum^+ y. h\text{-diff } i (y, x) * \text{indicator } X y * \text{indicator } (- X) x)$ 
  by(simp add: nn-integral-multc nn-integral-count-space-indicator)
also have  $\dots = (\sum^+ x \in UNIV. \sum^+ y. h\text{-diff } i (x, y) * \text{indicator } X x * \text{indicator } (- X) y)$ 

```

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

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

lemma *flow-cleanup*: — Lemma 5.4
 $\exists h \leq g. \text{flow } \Delta \ h \wedge \text{value-flow } \Delta \ h = \text{value-flow } \Delta \ g \wedge (\forall x. \ d\text{-IN } h \ x \leq \text{value-flow } \Delta \ h)$
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* :: $'v \text{ flow} \Rightarrow ('v, 'more) \text{ network-scheme}$
where *residual-network* $f =$
 $\langle \text{edge} = \lambda x \ y. \ \text{edge } \Delta \ x \ y \vee \text{edge } \Delta \ y \ x \wedge y \neq \text{source } \Delta,$
 $\text{capacity} = \lambda(x, y). \ \text{if } \text{edge } \Delta \ x \ y \text{ then } \text{capacity } \Delta \ (x, y) - f \ (x, y) \text{ else if } y =$
 $\text{source } \Delta \text{ then } 0 \text{ else } f \ (y, x),$
 $\text{source} = \text{source } \Delta, \text{sink} = \text{sink } \Delta, \dots = \text{network.more } \Delta \ \rangle$

lemma *residual-network-sel* [*simp*]:
 $\text{edge } (\text{residual-network } f) \ x \ y \longleftrightarrow \text{edge } \Delta \ x \ y \vee \text{edge } \Delta \ y \ x \wedge y \neq \text{source } \Delta$
 $\text{capacity } (\text{residual-network } f) \ (x, y) = (\text{if } \text{edge } \Delta \ x \ y \text{ then } \text{capacity } \Delta \ (x, y) - f \ (x, y) \text{ else if } y = \text{source } \Delta \text{ then } 0 \text{ else } f \ (y, x))$
 $\text{source } (\text{residual-network } f) = \text{source } \Delta$
 $\text{sink } (\text{residual-network } f) = \text{sink } \Delta$
 $\text{network.more } (\text{residual-network } f) = \text{network.more } \Delta$
by(*simp-all add: residual-network-def*)

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

by *auto*

lemma *vertices-residual-network* [*simp*]: $\text{vertex } (\text{residual-network } f) = \text{vertex } \Delta$
by(*auto simp add: vertex-def fun-eq-iff*)

inductive *wf-residual-network* :: *bool*

where $\llbracket \bigwedge x y. (x, y) \in \mathbf{E} \implies (y, x) \notin \mathbf{E}; (\text{source } \Delta, \text{sink } \Delta) \notin \mathbf{E} \rrbracket \implies \text{wf-residual-network}$

lemma *wf-residual-networkD*:

$\llbracket \text{wf-residual-network}; \text{edge } \Delta x y \rrbracket \implies \neg \text{edge } \Delta y x$

$\llbracket \text{wf-residual-network}; e \in \mathbf{E} \rrbracket \implies \text{prod.swap } e \notin \mathbf{E}$

$\llbracket \text{wf-residual-network}; \text{edge } \Delta (\text{source } \Delta) (\text{sink } \Delta) \rrbracket \implies \text{False}$

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* $?\Delta$)

proof

have *countable* (*converse* $\mathbf{E}$) **by** *simp*

then have *countable* $\{(x, y). (y, x) \in \mathbf{E} \wedge y \neq \text{source } \Delta\}$

by(*rule countable-subset[rotated]*) *auto*

then show *countable* $\mathbf{E}_{? \Delta}$ **unfolding** *E-residual-network* **by** *simp*

show *source* $?\Delta \neq \text{sink } ?\Delta$ **by** *simp*

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

show *capacity* $?\Delta e \neq \text{top}$ **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 $?\text{thesis}$

proof

show *d-IN* (*capacity* Δ'') $v \neq \top \vee \text{d-OUT}$ (*capacity* Δ'') $v \neq \top$ **if** $v \neq \text{sink}$

```

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

lemma  $\Delta''$ -wf-residual-network:
  assumes no-loop:  $\bigwedge x. \neg \text{edge } \Delta x x$ 
  shows  $\Delta''$ .wf-residual-network
by(auto simp add:  $\Delta''$ .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  $\Delta f$ 
  shows flow-attainability (residual-network  $f$ ) (is flow-attainability  $?\Delta$ )
proof –
  interpret res: countable-network residual-network  $f$  by(rule residual-countable-network[OF
assms])
  show ?thesis
  proof
    fix  $x$ 
    assume sink:  $x \neq \text{sink } ?\Delta$ 
    then consider (source)  $x = \text{source } \Delta \mid (\text{IN}) d\text{-IN } (\text{capacity } \Delta) x \neq \top \mid (\text{OUT})$ 
 $x \neq \text{source } \Delta d\text{-OUT } (\text{capacity } \Delta) x \neq \top$ 
    using finite-capacity[of  $x$ ] by auto
    then show  $d\text{-IN } (\text{capacity } ?\Delta) x \neq \top \vee d\text{-OUT } (\text{capacity } ?\Delta) x \neq \top$ 
    proof(cases)
      case source
      hence  $d\text{-IN } (\text{capacity } ?\Delta) x = 0$  by(simp add: d-IN-def source-in)
      thus ?thesis by simp
    next
      case IN
      have  $d\text{-IN } (\text{capacity } ?\Delta) x =$ 
         $(\sum^+ y. (\text{capacity } \Delta (y, x) - f (y, x)) * \text{indicator } \mathbf{E} (y, x) +$ 
         $(\text{if } x = \text{source } \Delta \text{ then } 0 \text{ else } f (x, y) * \text{indicator } \mathbf{E} (x, y)))$ 
      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  $\dots = (\sum^+ y. (\text{capacity } \Delta (y, x) - f (y, x)) * \text{indicator } \mathbf{E} (y, x))$ 
      +
       $(\sum^+ y. (\text{if } x = \text{source } \Delta \text{ then } 0 \text{ else } f (x, y) * \text{indicator } \mathbf{E} (x, y)))$ 
      (is  $- = ?in + ?out$ )
    qed
  qed

```

```

    by(subst nn-integral-add)(auto simp add: AE-count-space split: split-indicator
intro!: flowD-capacity[OF f])
    also have ... ≤ d-IN (capacity Δ) x + (if x = source Δ then 0 else d-OUT
f x) (is - ≤ ?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 ≠ source Δ x ≠ sink Δ
using sink by auto
    then have ?rest < ⊤
    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 =
      (∑+ 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 ... = (∑+ y. (capacity Δ (x, y) - f (x, y)) * indicator E (x, y))
+
      (∑+ 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 ... ≤ d-OUT (capacity Δ) x + d-IN f x (is - ≤ ?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 + ... ≤ ?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 ¬ edge ?Δ x x for x by(simp add: no-loop)
  show ¬ edge ?Δ x (source ?Δ) for x by(simp add: source-in)
qed
qed
end

definition plus-flow :: ('v, 'more) graph-scheme ⇒ 'v flow ⇒ 'v flow ⇒ 'v flow

```

```

(infixr <⊕1> 65)
where plus-flow  $G f g = (\lambda(x, y). \text{ if edge } G x y \text{ then } f(x, y) + g(x, y) - g(y, x) \text{ else } 0)$ 

lemma plus-flow-simps [simp]: fixes  $G$  (structure) shows
   $(f \oplus g)(x, y) = (\text{ if edge } G x y \text{ then } f(x, y) + g(x, y) - g(y, x) \text{ else } 0)$ 
by(simp add: plus-flow-def)

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

lemma
  fixes  $\Delta$  (structure)
  assumes f-outside:  $\bigwedge e. e \notin \mathbf{E} \implies f e = 0$ 
  and g-le-f:  $\bigwedge x y. \text{ edge } \Delta x y \implies g(y, x) \leq f(x, y)$ 
  shows OUT-plus-flow:  $d\text{-IN } g x \neq \text{top} \implies d\text{-OUT } (f \oplus g) x = d\text{-OUT } f x +$ 
 $(\sum^+_{y \in \text{UNIV}. g(x, y) * \text{indicator } \mathbf{E}(x, y)) - (\sum^+ y. g(y, x) * \text{indicator } \mathbf{E}(x,$ 
 $y))$ 
    (is -  $\implies ?\text{OUT}$  is -  $\implies - = - + ?g\text{-out} - ?g\text{-out}'$ )
    and IN-plus-flow:  $d\text{-OUT } g x \neq \text{top} \implies d\text{-IN } (f \oplus g) x = d\text{-IN } f x + (\sum^+_{y \in \text{UNIV}. g(y, x) * \text{indicator } \mathbf{E}(y, x)) - (\sum^+ y. g(x, y) * \text{indicator } \mathbf{E}(y, x))$ 
    (is -  $\implies ?\text{IN}$  is -  $\implies - = - + ?g\text{-in} - ?g\text{-in}'$ )
proof -
  assume d-IN  $g x \neq \text{top}$ 
  then have finite1:  $(\sum^+ y. g(y, x) * \text{indicator } A(f y)) \neq \text{top}$  for  $A f$ 
    by(rule neq-top-trans)(auto split: split-indicator simp add: d-IN-def intro!: nn-integral-mono)

  have d-OUT  $(f \oplus g) x = (\sum^+ y. (g(x, y) + (f(x, y) - g(y, x))) * \text{indicator } \mathbf{E}(x, y))$ 
  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 ... =  $?g\text{-out} + (\sum^+ y. (f(x, y) - g(y, x)) * \text{indicator } \mathbf{E}(x, y))$ 
    (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  $?rest = (\sum^+ y. f(x, y) * \text{indicator } \mathbf{E}(x, y)) - ?g\text{-out}'$  (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  $?f = d\text{-OUT } f x$  unfolding d-OUT-def using f-outside
  by(auto intro!: nn-integral-cong split: split-indicator)
  also have  $(\sum^+ y. g(x, y) * \text{indicator } \mathbf{E}(x, y)) + (d\text{-OUT } f x - (\sum^+ y. g(y, x) * \text{indicator } \mathbf{E}(x, y))) =$ 
 $d\text{-OUT } f x + ?g\text{-out} - ?g\text{-out}'$ 
  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* $g\ x \neq \text{top}$
 then have *finite2*: $(\sum^+ y. g\ (x, y) * \text{indicator}\ A\ (f\ y)) \neq \text{top}$ **for** *A* *f*
 by(*rule* *neq-top-trans*)(*auto* *split*: *split-indicator simp add: d-OUT-def intro!*:
nn-integral-mono)

 have *d-IN* $(f \oplus g)\ x = (\sum^+ y. (g\ (y, x) + (f\ (y, x) - g\ (x, y))) * \text{indicator}\ \mathbf{E}\ (y, x))$
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 $\dots = (\sum^+ y \in \text{UNIV}. g\ (y, x) * \text{indicator}\ \mathbf{E}\ (y, x)) + (\sum^+ y. (f\ (y, x) - g\ (x, y)) * \text{indicator}\ \mathbf{E}\ (y, x))$
 (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* = $(\sum^+ y. f\ (y, x) * \text{indicator}\ \mathbf{E}\ (y, x)) - ?g\text{-in}'$
 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 $(\sum^+ y. f\ (y, x) * \text{indicator}\ \mathbf{E}\ (y, x)) = d\text{-IN}\ f\ x$
unfolding *d-IN-def* **using** *f-outside* **by**(*auto* *intro!*: *nn-integral-cong split:*
split-indicator)
 also have $(\sum^+ y. g\ (y, x) * \text{indicator}\ \mathbf{E}\ (y, x)) + (d\text{-IN}\ f\ x - (\sum^+ y. g\ (x, y) * \text{indicator}\ \mathbf{E}\ (y, x))) =$
 $d\text{-IN}\ f\ x + ?g\text{-in} - ?g\text{-in}'$
 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* $\Delta\ f$
 and *g*: *flow* (*residual-network* *f*) *g*
 shows *d-IN* $(f \oplus g)\ x \leq d\text{-IN}\ f\ x + d\text{-IN}\ g\ x$

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 $\dots = d\text{-IN}\ f\ x + d\text{-IN}\ g\ x$
 by(*subst* *nn-integral-add*)(*simp*-*all* *add: d-IN-def*)
finally show ?*thesis* .
qed

lemma *scale-flow*:

assumes *f*: *flow* $\Delta\ f$
 and *c*: $c \leq 1$
 shows *flow* $\Delta\ (\lambda e. c * f\ e)$

```

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

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

lemma value-flow:
  assumes f: flow  $\Delta f$ 
  and source-out:  $\bigwedge y. \text{edge } \Delta (\text{source } \Delta) y \longleftrightarrow y = x$ 
  shows  $\text{value-flow } \Delta f = f (\text{source } \Delta, x)$ 
proof -
  have  $\text{value-flow } \Delta f = (\sum^+ y \in \text{OUT } (\text{source } \Delta). f (\text{source } \Delta, y))$ 
  by(rule d-OUT-alt-def)(simp add: flowD-outside[OF f])
  also have  $\dots = (\sum^+ y. f (\text{source } \Delta, y) * \text{indicator } \{x\} y)$ 
  by(subst nn-integral-count-space-indicator)(auto intro!: nn-integral-cong split:
split-indicator simp add: outgoing-def source-out)
  also have  $\dots = f (\text{source } \Delta, x)$  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  $\Delta f$ 
  and g: flow (residual-network f) g
  shows  $\text{value-flow } \Delta (f \oplus g) = \text{value-flow } \Delta f + \text{value-flow } \Delta g$ 
proof -
  interpret RES: countable-network residual-network f using wf f by(rule resid-
ual-countable-network)
  have  $\text{value-flow } \Delta (f \oplus g) = (\sum^+ y. f (\text{source } \Delta, y) + g (\text{source } \Delta, y))$ 
  unfolding d-OUT-def by(rule nn-integral-cong)(simp add: flowD-outside[OF
f] RES.flowD-outside[OF g] source-in)
  also have  $\dots = \text{value-flow } \Delta f + \text{value-flow } \Delta g$  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 $\in$ E
 hence (f $\oplus$ g) e = f e + g e - g (prod.swap e) by(cases e) simp
 also have ... $\leq$ f e + g e - 0 by(rule ennreal-minus-mono) simp-all
 also have ... $\leq$ 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 ... = 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: ($\sum^+ y. g (y, x) * \text{indicator } A (f y)$) $\neq \top$ 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: ($\sum^+ y. g (x, y) * \text{indicator } A (f y)$) $\neq \top$ 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 + ($\sum^+ y. g (x, y) * \text{indicator } \mathbf{E} (x, y)$) -
 ($\sum^+ y. g (y, x) * \text{indicator } \mathbf{E} (x, y)$)
 (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 = ($\sum^+ y. g (x, y) * \text{indicator } (- \mathbf{E}) (y, x)$)
proof -
 have ($\sum^+ y. g (x, y) * \text{indicator } (- \mathbf{E}) (y, x)$) =
 ($\sum^+ y. g (x, y) * \text{indicator } \mathbf{E} (x, y)$) + ($\sum^+ y. g (x, y) * \text{indicator } (-$

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

definition *minus-flow* :: 'v flow $\Rightarrow$ 'v flow $\Rightarrow$ 'v flow (**infixl** $\ominus$ 65)

where

$f \ominus g = (\lambda(x, y). \text{ if edge } \Delta x y \text{ then } f(x, y) - g(x, y) \text{ else if edge } \Delta y x \text{ then } g(y, x) - f(y, x) \text{ else } 0)$

lemma *minus-flow-simps* [*simp*]:

$(f \ominus g)(x, y) = (\text{ if edge } \Delta x y \text{ then } f(x, y) - g(x, y) \text{ else if edge } \Delta y x \text{ then } g(y, x) - f(y, x) \text{ else } 0)$

by(*simp add: minus-flow-def*)

lemma *minus-flow*:

assumes *wf*: *wf-residual-network*

and *f*: *flow* Δf

and *g*: *flow* Δg

and *value-le*: *value-flow* $\Delta g \leq \text{value-flow } \Delta f$

and *f-finite*: $f(\text{source } \Delta, x) \neq \top$

and *source-out*: $\bigwedge y. \text{ edge } \Delta (\text{source } \Delta) y \longleftrightarrow y = x$

shows *flow* (*residual-network* *g*) ($f \ominus g$) (**is flow** $? \Delta ?f$)

proof

show $?f e \leq \text{capacity } ? \Delta e$ **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 $x \neq \text{source } ? \Delta x \neq \text{sink } ? \Delta$

hence *x*: $x \neq \text{source } \Delta x \neq \text{sink } \Delta$ **by** *simp-all*

have *finite-f-in*: $(\sum^+ y. f(y, x) * \text{indicator } A y) \neq \text{top}$ **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*: $(\sum^+ y. f(x, y) * \text{indicator } A y) \neq \text{top}$ **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*]: $f(x, y) \neq \text{top} \wedge f(y, x) \neq \text{top}$ **for** *y*

using *finite-f-in*[*of* $\{y\}$] *finite-f-out*[*of* $\{y\}$] **by** *auto*

have *finite-g-in*: $(\sum^+ y. g(y, x) * \text{indicator } A y) \neq \text{top}$ **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*: $(\sum^+ y. g(x, y) * \text{indicator } A y) \neq \text{top}$ **for** *A*

using *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 *finite-g*[*simp*]: $g(x, y) \neq \text{top} \wedge g(y, x) \neq \text{top}$ **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 E (x, y) *
indicator {y. g (x, y)  $\leq$  f (x, y)} y) +
( $\sum^+$  y. (g (y, x) - f (y, x)) * indicator 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 fi-
nite-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 ... =  $d\text{-OUT } f \ x + (?g\text{-less-}f - ?f\text{-out-less}) - d\text{-OUT } g \ x + (?g\text{-in} - ?f\text{-in})$ 
by (subst add-diff-eq-ennreal[symmetric])
      (auto intro!: nn-integral-mono split: split-indicator)
also have ... =  $(?g\text{-in} - ?f\text{-in}) + ((?g\text{-less-}f - ?f\text{-out-less}) + d\text{-OUT } f \ x - d\text{-OUT } g \ x)$ 
by (simp add: ac-simps)
also have ... =  $((?g\text{-in} - ?f\text{-in}) + ((?g\text{-less-}f - ?f\text{-out-less}) + d\text{-OUT } f \ x)) - d\text{-OUT } g \ x$ 
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 ... =  $((?g\text{-less-}f - ?f\text{-out-less}) + (d\text{-OUT } f \ x + (?g\text{-in} - ?f\text{-in}))) - d\text{-OUT } g \ x$ 
by (simp add: ac-simps)
also have ... =  $((?g\text{-less-}f - ?f\text{-out-less}) + (d\text{-IN } f \ x + (?g\text{-in} - ?f\text{-in}))) - d\text{-IN } g \ x$ 
unfolding flowD-KIR[OF f x] flowD-KIR[OF g x] ..
also have ... =  $(?g\text{-less-}f - ?f\text{-out-less}) + ((d\text{-IN } f \ x + (?g\text{-in} - ?f\text{-in})) - d\text{-IN } g \ x)$ 
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 ... =  $(?g\text{-less-}f - ?f\text{-out-less}) + (d\text{-IN } f \ x + ?g\text{-in} - d\text{-IN } g \ x - ?f\text{-in})$ 
by (subst (2) add-diff-eq-ennreal) (auto intro!: nn-integral-mono split: split-indicator
simp: diff-diff-commute-ennreal)
also have ... =  $(?g\text{-less-}f - ?f\text{-out-less}) + (d\text{-IN } f \ x - (d\text{-IN } g \ x - ?g\text{-in} - ?f\text{-in}))$ 
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 ... =  $(d\text{-IN } f \ x - ?f\text{-in}) - (d\text{-IN } g \ x - ?g\text{-in}) + (?g\text{-less-}f - ?f\text{-out-less})$ 
by (simp add: ac-simps diff-diff-commute-ennreal)
also have  $?g\text{-less-}f - ?f\text{-out-less} = (\sum^+ y. (g \ (x, y) - f \ (x, y)) * \text{indicator } \{y. f \ (x, y) < g \ (x, y)\} \ y)$  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 ... =  $(\sum^+ y. (g \ (x, y) - f \ (x, y)) * \text{indicator } \mathbf{E} \ (x, y) * \text{indicator}$ 

```

```

{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:  $\bigwedge y. \text{edge } \Delta (\text{source } \Delta) y \longleftrightarrow y = x$ 
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  $\Delta$  f - value-flow  $\Delta$  g using f g source-out by(simp
add: value-flow)
    finally show ?value .
qed

context
  fixes  $\alpha$ 
  defines  $\alpha \equiv (SUP\ g \in \{g. \text{flow } \Delta\ g\}. \text{value-flow } \Delta\ g)$ 
begin

lemma flow-by-value:
  assumes  $v < \alpha$ 
  and real[rule-format]:  $\forall f. \alpha = \top \longrightarrow \text{flow } \Delta\ f \longrightarrow \text{value-flow } \Delta\ f < \alpha$ 
  obtains f where flow  $\Delta$  f value-flow  $\Delta$  f = v
proof -
  have  $\alpha\text{-pos}: \alpha > 0$  using assms by (auto simp add: zero-less-iff-neq-zero)
  from  $\langle v < \alpha \rangle$  obtain f where f: flow  $\Delta$  f and v:  $v < \text{value-flow } \Delta\ f$ 
    unfolding  $\alpha\text{-def}$  less-SUP-iff by blast
  have [simp]: value-flow  $\Delta$  f  $\neq \top$ 
  proof
    assume val: value-flow  $\Delta$  f =  $\top$ 
    from f have value-flow  $\Delta$  f  $\leq \alpha$  unfolding  $\alpha\text{-def}$  by (blast intro: SUP-upper2)
    with val have  $\alpha = \top$  by (simp add: top-unique)
    from real[OF this f] val show False by simp
  qed
  let ?f =  $\lambda e. (v / \text{value-flow } \Delta\ f) * f\ e$ 
  note f
  moreover
  have *:  $0 < \text{value-flow } \Delta\ f$ 
    using  $\langle v < \text{value-flow } \Delta\ f \rangle$  by (auto simp add: zero-less-iff-neq-zero)
  then have  $v / \text{value-flow } \Delta\ f \leq 1$  using v
    by (auto intro!: divide-le-posI-ennreal)
  ultimately have flow  $\Delta$  ?f by (rule scale-flow)
  moreover {
    have value-flow  $\Delta$  ?f =  $v * (\text{value-flow } \Delta\ f / \text{value-flow } \Delta\ f)$ 
      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:  $\bigwedge y. \text{edge } \Delta\ (\text{source } \Delta)\ y \longleftrightarrow y = x$ 
  and nontrivial:  $\mathbf{V} - \{\text{source } \Delta, \text{sink } \Delta\} \neq \{\}$ 
  and real:  $\alpha = \text{ennreal } \alpha'$  and  $\alpha'\text{-nonneg}$ [simp]:  $0 \leq \alpha'$ 
  shows  $\exists f. \text{flow } \Delta\ f \wedge \text{value-flow } \Delta\ f = \alpha \wedge (\forall x. d\text{-IN } f\ x \leq \text{value-flow } \Delta\ f)$ 
proof -
  have  $\alpha'\text{-not-neg}$ [simp]:  $\neg \alpha' < 0$ 

```

```

using  $\alpha'$ -nonneg by linarith

let  $?v = \lambda i. (1 - (1 / 2) ^ i) * \alpha$ 
let  $?v-r = \lambda i. \text{ennreal } ((1 - (1 / 2) ^ i) * \alpha')$ 
have  $v\text{-eq}: ?v\ i = ?v-r\ i$  for  $i$ 
by (auto simp: real ennreal-mult power-le-one ennreal-lessI ennreal-minus[symmetric]
      ennreal-power[symmetric] divide-ennreal-def)
have  $\exists f. \text{flow } \Delta f \wedge \text{value-flow } \Delta f = ?v\ i$  for  $i :: \text{nat}$ 
proof(cases  $\alpha = 0$ )
  case True thus  $?thesis$  by(auto intro!: exI[where x= $\lambda-. 0$ ])
next
  case False
  then have  $?v\ i < \alpha$ 
  unfolding  $v\text{-eq}$  by (auto simp: real field-simps intro!: ennreal-lessI) (simp-all
add: less-le)
  then obtain  $f$  where  $\text{flow } \Delta f$  and  $\text{value-flow } \Delta f = ?v\ i$ 
  by(rule flow-by-value)(simp add: real)
  thus  $?thesis$  by blast
qed
then obtain  $f\text{-aux}$  where  $f\text{-aux}: \bigwedge i. \text{flow } \Delta (f\text{-aux}\ i)$ 
and  $\text{value-aux}: \bigwedge i. \text{value-flow } \Delta (f\text{-aux}\ i) = ?v-r\ i$ 
unfolding  $v\text{-eq}$  by moura

define  $f\text{-i}$  where  $f\text{-i} = \text{rec-nat } (\lambda-. 0) (\lambda i\ f.i.$ 
   $\text{let } g = f\text{-aux } (\text{Suc } i) \ominus f\text{-i};$ 
   $k\text{-i} = \text{SOME } k. k \leq g \wedge \text{flow } (\text{residual-network } f\text{-i})\ k \wedge \text{value-flow } (\text{residual-network}$ 
 $f\text{-i})\ k = \text{value-flow } (\text{residual-network } f\text{-i})\ g \wedge (\forall x. d\text{-IN } k\ x \leq \text{value-flow } (\text{residual-network}$ 
 $f\text{-i})\ k)$ 
   $\text{in } f\text{-i} \oplus k\text{-i})$ 
  let  $?P = \lambda i\ k. k \leq f\text{-aux } (\text{Suc } i) \ominus f\text{-i}\ i \wedge \text{flow } (\text{residual-network } (f\text{-i}\ i))\ k \wedge$ 
 $\text{value-flow } (\text{residual-network } (f\text{-i}\ i))\ k = \text{value-flow } (\text{residual-network } (f\text{-i}\ i))\ (f\text{-aux}$ 
 $(\text{Suc } i) \ominus f\text{-i}\ i) \wedge (\forall x. d\text{-IN } k\ x \leq \text{value-flow } (\text{residual-network } (f\text{-i}\ i))\ k)$ 
  define  $k\text{-i}$  where  $k\text{-i}\ i = \text{Eps } (?P\ i)$  for  $i$ 

have  $f\text{-i-simps } [simp]: f\text{-i}\ 0 = (\lambda-. 0)\ f\text{-i}\ (\text{Suc } i) = f\text{-i}\ i \oplus k\text{-i}\ i$  for  $i$ 
by(simp-all add: f-i-def Let-def k-i-def)

have  $k\text{-i}: \text{flow } (\text{residual-network } (f\text{-i}\ i))\ (k\text{-i}\ i) \text{ (is } ?k\text{-i)}$ 
and  $\text{value-k-i}: \text{value-flow } (\text{residual-network } (f\text{-i}\ i))\ (k\text{-i}\ i) = \text{value-flow } (\text{residual-network}$ 
 $(f\text{-i}\ i))\ (f\text{-aux } (\text{Suc } i) \ominus f\text{-i}\ i) \text{ (is } ?\text{value-k-i})$ 
and  $\text{IN-k-i}: d\text{-IN } (k\text{-i}\ i)\ x \leq \text{value-flow } (\text{residual-network } (f\text{-i}\ i))\ (k\text{-i}\ i) \text{ (is}$ 
 $?IN\text{-k-i})$ 
and  $\text{value-diff}: \text{value-flow } (\text{residual-network } (f\text{-i}\ i))\ (f\text{-aux } (\text{Suc } i) \ominus f\text{-i}\ i) =$ 
 $\text{value-flow } \Delta (f\text{-aux } (\text{Suc } i)) - \text{value-flow } \Delta (f\text{-i}\ i) \text{ (is } ?\text{value-diff})$ 
if  $\text{flow-network } \Delta (f\text{-i}\ i)$  and  $\text{value-f-i}: \text{value-flow } \Delta (f\text{-i}\ i) = \text{value-flow } \Delta$ 
 $(f\text{-aux}\ i)$  for  $i\ x$ 
proof -
  let  $?RES = \text{residual-network } (f\text{-i}\ i)$ 
  interpret  $fn: \text{flow-network } \Delta f\text{-i}\ i$  by(rule that)

```

```

interpret RES: flow-attainability ?RES using wf fn.g by(rule residual-flow-attainability)
have le: value-flow  $\Delta$  (f-i i)  $\leq$  value-flow  $\Delta$  (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-locale) 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  $\Delta$  (f-i i)
  and value-f-i: value-flow  $\Delta$  (f-i i) = value-flow  $\Delta$  (f-aux i)
  and d-IN-i: d-IN (f-i i)  $x \leq$  value-flow  $\Delta$  (f-i i) for i x
proof(induction i)
case 0
{ case 1 show ?case using nontrivial by(unfold-locale)(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  $\Delta$  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  $\Delta$  (f-i i)  $\leq$  value-flow  $\Delta$  (f-aux (Suc i))
  unfolding value-aux Suc.IH(2) v-eq using  $\alpha'$ -nonneg by(intro ennreal-leI)(simp
add: real field-simps)
{ case 1 show ?case unfolding f-i-simps
proof
  show flow  $\Delta$  (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  $\Delta$  (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  $\Delta$  (f-aux

```

```

(Suc i)) - value-flow Δ (f-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-IN (f-i i ⊕ k-i i) x ≤ d-IN (f-i i) x + d-IN (k-i i) x
      using fn.g k-i by(rule d-IN-plus-flow[OF wf])
    also have ... ≤ value-flow Δ (f-i i) + d-IN (k-i i) x using Suc.IH(3) by(rule
      add-right-mono)
    also have ... ≤ value-flow Δ (f-i i) + value-flow ?RES (k-i i) using d-IN-k-i
      by(rule add-left-mono)
    also have ... = value-flow Δ (f-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 ∃ x ≥ 0. f-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': ∧ i e. f-i i e = ennreal (f-i' i e) and [simp]: ∧ i e.
    0 ≤ f-i' i e
    by metis

  { fix i e
    obtain x y :: 'v 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) by(rule IN-k-i)
    also have ... < ⊤ 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 ∃ x ≥ 0. k-i i e = ennreal x
      by(cases k-i i e)(auto simp add: e) }
  then obtain k-i' where k-i': ∧ i e. k-i i e = ennreal (k-i' i e) and k-i'-nonneg[simp]:
    ∧ i e. 0 ≤ k-i' i e
    by metis

  have wf-k: (x, y) ∈ E ⟹ k-i i (y, x) ≤ f-i i (x, y) + k-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-i' 0 = (λ-. 0) using f-i-simps(1) by (simp del: f-i-simps
    add: fun-eq-iff f-i')

  have f-i'-Suc[simp]: f-i' (Suc i) e = (if e ∈ E then f-i' i e + (k-i' i e - k-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 =  $\lambda i x y. (k-i' i (x, y) - k-i' i (y, x)) * \text{indicator } \mathbf{E} (x, y)$ 
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 ( $\lambda i. |?k i x y|$ ) for  $x y$ 
proof (rule summable-rabs-comparison-test)
  show  $\exists N. \forall i \geq N. |?k i x y| \leq \alpha' * (1 / 2) ^ i$ 
  proof (intro exI allI impI)
    fix i have  $|?k i x y| \leq k-i' i (x, y) + k-i' i (y, x)$ 
    by (auto intro!: abs-triangle-ineq4 [THEN order-trans] split: split-indicator)
    also have  $\dots \leq \alpha' * (1 / 2) ^ i$ 
    using k-i'-le[of i (x, y)] k-i'-le[of i (y, x)]  $\alpha'$ -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  $|?k i x y| \leq \alpha' * (1 / 2) ^ i$ 
  by simp
qed
show summable ( $\lambda i. \alpha' * (1 / 2) ^ i$ )
by (rule summable-mult complete-algebra-summable-geometric)+ simp
qed
hence summable-k: summable ( $\lambda i. ?k i x y$ ) for  $x y$  by (auto intro: summable-norm-cancel)

have suminf:  $(\sum i. (k-i' i (x, y) - k-i' i (y, x)) * \text{indicator } \mathbf{E} (x, y)) = f' (x,$ 
 $y)$  for  $x y$ 
by (rule LIMSEQ-unique[OF summable-LIMSEQ])(simp-all add: sum-i'[symmetric]
f' summable-k)

have flow: flow  $\Delta f'$ 
proof
  fix e
  have  $f' e \leq \text{Sup } \{.. \text{capacity } \Delta e\}$  using - f
  by (rule Sup-lim)(simp add: flowD-capacity[OF fn.g])
  then show  $f' e \leq \text{capacity } \Delta e$  by simp
next
  fix x
  assume  $x: x \neq \text{source } \Delta x \neq \text{sink } \Delta$ 

  have integrable-f-i: integrable (count-space UNIV) ( $\lambda y. f-i' i (x, y)$ ) 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) ( $\lambda y. f-i' i (y, x)$ ) 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:  $(\sum^+ y. \text{norm } (?k i x y)) \leq \alpha' / 2 ^ i$  (is ?thesis1)
  and integral-k'-bounded:  $(\sum^+ y. \text{norm } (?k i y x)) \leq \alpha' / 2 ^ i$  (is ?thesis2)
  for i

```

```

proof -
  define b where b = ( $\sum^+ y. k\text{-}i\ i\ (x, y) + k\text{-}i\ i\ (y, x)$ )
  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 ... ≤  $\alpha' / 2^i$  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 ( $\sum^+ y \in UNIV. \text{norm } (?k\ i\ x\ y)$ ) ≤ b and ( $\sum^+ y. \text{norm } (?k\ i\ y\ x)$ ) ≤ 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) ( $\lambda y. ?k\ i\ x\ y$ )
  and integrable-k': integrable (count-space UNIV) ( $\lambda y. ?k\ i\ y\ x$ ) 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 ( $\lambda i. \int y. |?k\ i\ x\ y| \partial \text{count-space UNIV}$ )
proof(rule summable-comparison-test)
  have  $|\int y. |?k\ i\ x\ y| \partial \text{count-space UNIV}| \leq \alpha' * (1 / 2)^i$  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  $\exists N. \forall i \geq N. \text{norm } (\int y. |?k\ i\ x\ y| \partial \text{count-space UNIV}) \leq \alpha' * (1 / 2)^i$ 
    by auto
  show summable ( $\lambda i. \alpha' * (1 / 2)^i$ )
    by(rule summable-mult complete-algebra-summable-geometric)+ simp
qed
have summable'-k': summable ( $\lambda i. \int y. |?k\ i\ y\ x| \partial \text{count-space UNIV}$ )
proof(rule summable-comparison-test)
  have  $|\int y. |?k\ i\ y\ x| \partial \text{count-space UNIV}| \leq \alpha' * (1 / 2)^i$  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  $\exists N. \forall i \geq N. \text{norm } (\int y. |?k\ i\ y\ x| \partial \text{count-space UNIV}) \leq \alpha' * (1 / 2)^i$ 
    by auto
  show summable ( $\lambda i. \alpha' * (1 / 2)^i$ )
    by(rule summable-mult complete-algebra-summable-geometric)+ simp
qed

```

have $(\lambda i. \int y. ?k \ i \ x \ y \ \partial \text{count-space UNIV}) \text{ sums } \int y. (\sum i. ?k \ i \ x \ y)$
 $\partial \text{count-space UNIV}$
using *integrable-k* **by**(rule *sums-integral*)(simp-all add: *summable-nk summable'-k*)
also have $\dots = \int y. f'(x, y) \ \partial \text{count-space UNIV}$ **by**(rule *Bochner-Integration.integral-cong[OF refl]*)(rule *suminf*)
finally have $(\lambda i. \sum j < i. \int y. ?k \ j \ x \ y \ \partial \text{count-space UNIV}) \longrightarrow \dots$ **un-**
folding *sums-def* .
also have $(\lambda i. \sum j < i. \int y. ?k \ j \ x \ y \ \partial \text{count-space UNIV}) = (\lambda i. \int y. f-i' \ i \ (x,$
 $y) \ \partial \text{count-space UNIV})$
unfolding *sum-i'* **by**(rule *ext Bochner-Integration.integral-sum[symmetric]*
integrable-k)+
finally have $(\lambda i. \text{ennreal} (\int y. f-i' \ i \ (x, y) \ \partial \text{count-space UNIV})) \longrightarrow \text{ennreal}$
 $(\int y. f'(x, y) \ \partial \text{count-space UNIV})$ **by** *simp*
also have $(\lambda i. \text{ennreal} (\int y. f-i' \ i \ (x, y) \ \partial \text{count-space UNIV})) = (\lambda i. d-OUT$
 $(f-i \ i) \ x)$
unfolding *d-OUT-def f-i'* **by**(rule *ext nn-integral-eq-integral[symmetric]* *inte-*
grable-f-i)+ *simp*
also have $\text{ennreal} (\int y. f'(x, y) \ \partial \text{count-space UNIV}) = d-OUT \ f' \ x$
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 $(\lambda i. d-OUT \ (f-i \ i) \ x) = (\lambda i. d-IN \ (f-i \ i) \ x)$
using *flowD-KIR[OF fn.g x]* **by**(*simp*)
finally have *: $(\lambda i. d-IN \ (f-i \ i) \ x) \longrightarrow d-OUT \ (\lambda x. \text{ennreal} \ (f' \ x)) \ x$.

have $(\lambda i. \int y. ?k \ i \ y \ x \ \partial \text{count-space UNIV}) \text{ sums } \int y. (\sum i. ?k \ i \ y \ x)$
 $\partial \text{count-space UNIV}$
using *integrable-k'* **by**(rule *sums-integral*)(simp-all add: *summable-nk summable'-k'*)
also have $\dots = \int y. f'(y, x) \ \partial \text{count-space UNIV}$ **by**(rule *Bochner-Integration.integral-cong[OF refl]*)(rule *suminf*)
finally have $(\lambda i. \sum j < i. \int y. ?k \ j \ y \ x \ \partial \text{count-space UNIV}) \longrightarrow \dots$ **un-**
folding *sums-def* .
also have $(\lambda i. \sum j < i. \int y. ?k \ j \ y \ x \ \partial \text{count-space UNIV}) = (\lambda i. \int y. f-i' \ i \ (y,$
 $x) \ \partial \text{count-space UNIV})$
unfolding *sum-i'* **by**(rule *ext Bochner-Integration.integral-sum[symmetric]*
integrable-k')+
finally have $(\lambda i. \text{ennreal} (\int y. f-i' \ i \ (y, x) \ \partial \text{count-space UNIV})) \longrightarrow \text{ennreal}$
 $(\int y. f'(y, x) \ \partial \text{count-space UNIV})$ **by** *simp*
also have $(\lambda i. \text{ennreal} (\int y. f-i' \ i \ (y, x) \ \partial \text{count-space UNIV})) = (\lambda i. d-IN \ (f-i$
 $i) \ x)$
unfolding *d-IN-def f-i'* **by**(rule *ext nn-integral-eq-integral[symmetric]* *inte-*
grable-f-i')+ *simp*
also have $\text{ennreal} (\int y. f'(y, x) \ \partial \text{count-space UNIV}) = d-IN \ f' \ x$
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 $d-OUT \ f' \ x = d-IN \ f' \ x$ **using** * **by**(blast intro: *LIMSEQ-unique*)
qed
moreover


```

{ have incseq ( $\lambda i.$  value-flow  $\Delta$  (f-i i))
  by(rule incseq-SucI)(simp add: value-aux value-f-i real field-simps  $\alpha'$ -nonneg
ennreal-leI del: f-i-simps)
  then have ( $\lambda i.$  value-flow  $\Delta$  (f-i i))  $\longrightarrow$  (SUP i. value-flow  $\Delta$  (f-i i)) by(rule
LIMSEQ-SUP)
  also have (SUP i. value-flow  $\Delta$  (f-i i)) =  $\alpha$ 
  proof -
    have  $\alpha -$  (SUP i. value-flow  $\Delta$  (f-i i)) = (INF i.  $\alpha -$  value-flow  $\Delta$  (f-i i))
      by(simp add: ennreal-SUP-const-minus real)
    also have  $\alpha -$  value-flow  $\Delta$  (f-i i) =  $\alpha' / 2^i$  for i
    by(simp add: value-f-i value-aux real ennreal-minus-if field-simps mult-le-cancel-left1)
    hence (INF i.  $\alpha -$  value-flow  $\Delta$  (f-i i)) = (INF i. ennreal ( $\alpha' / 2^i$ ))
      by(auto intro: INF-cong)
    also have ... = 0
  proof(rule LIMSEQ-unique)
    show ( $\lambda i.$   $\alpha' / 2^i$ )  $\longrightarrow$  (INF i. ennreal ( $\alpha' / 2^i$ ))
      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  $\Delta$  (f-i i)) =  $\alpha$ 
      apply (intro antisym)
      apply (auto simp:  $\alpha$ -def intro!: SUP-mono fn.g) []
      apply (rule ennreal-minus-eq-0)
      apply assumption
    done
  qed
  also have ( $\lambda i.$  value-flow  $\Delta$  (f-i i))  $\longrightarrow$  value-flow  $\Delta$  f'
    by(simp add: value-flow[OF flow source-out] value-flow[OF fn.g source-out]
f)
  ultimately have value-flow  $\Delta$  f' =  $\alpha$  by(blast intro: LIMSEQ-unique) }
  note value-f = this
  moreover {
    fix x
    have d-IN f' x =  $\int^+ y.$  liminf ( $\lambda i.$  f-i i (y, x))  $\partial$ count-space UNIV unfolding
d-IN-def using f
      by(simp add: tendsto-iff-Liminf-eq-Limsup)
    also have ...  $\leq$  liminf ( $\lambda i.$  d-IN (f-i i) x) unfolding d-IN-def
      by(rule nn-integral-liminf)(simp-all add:)
    also have ...  $\leq$  liminf ( $\lambda i.$   $\alpha$ ) using d-IN-i[of - x] fn.g
      by(auto intro!: Liminf-mono SUP-upper2 eventually-sequentiallyI simp add:
 $\alpha$ -def)
    also have ... = value-flow  $\Delta$  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:  $\bigwedge y.$  edge  $\Delta$  (source  $\Delta$ ) y  $\longleftrightarrow$  y = x

```

and *nontrivial*: $\mathbf{E} \neq \{\}$
and *real*: $\alpha = \text{ennreal } \alpha'$ **and** *nn[simp]*: $0 \leq \alpha'$
shows $\exists f. \text{flow } \Delta f \wedge \text{value-flow } \Delta f = \alpha \wedge (\forall x. d\text{-IN } f x \leq \text{value-flow } \Delta f)$
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'') $y \longleftrightarrow y = \text{Edge } (\text{source } \Delta) x$ **for** y
by(*auto simp add*: *source-out*)
have *nontrivial'*: $\mathbf{V}_{\Delta''} - \{\text{source } \Delta'', \text{sink } \Delta''\} \neq \{\}$ **using** *nontrivial* **by**(*auto simp add*: $\mathbf{V}-\Delta''$)

have $(\text{SUP } g \in \{g. \text{flow } \Delta'' g\}. \text{value-flow } \Delta'' g) = (\text{SUP } g \in \{g. \text{flow } \Delta g\}. \text{value-flow } \Delta g)$ **(is ?lhs = ?rhs)**
proof(*intro antisym SUP-least; unfold mem-Collect-eq*)
fix g
assume $g: \text{flow } \Delta'' g$
hence $\text{value-flow } \Delta'' g = \text{value-flow } \Delta (\text{collect } g)$ **by**(*simp add*: *value-collect*)
also { **from** g **have** $\text{flow } \Delta (\text{collect } g)$ **by** *simp* }
then **have** $\dots \leq ?rhs$ **by**(*blast intro*: *SUP-upper2*)
finally **show** $\text{value-flow } \Delta'' g \leq \dots$.
next
fix g
assume $g: \text{flow } \Delta g$
hence $\text{value-flow } \Delta g = \text{value-flow } \Delta'' (\text{split } g)$ **by** *simp*
also { **from** g **have** $\text{flow } \Delta'' (\text{split } g)$ **by** *simp* }
then **have** $\dots \leq ?lhs$ **by**(*blast intro*: *SUP-upper2*)
finally **show** $\text{value-flow } \Delta g \leq ?lhs$.
qed
with *real* **have** *eq*: $(\text{SUP } g \in \{g. \text{flow } \Delta'' g\}. \text{value-flow } \Delta'' g) = \text{ennreal } \alpha'$
by(*simp add*: $\alpha\text{-def}$)
from $\Delta''.\text{ex-max-flow}'[\text{OF } \text{wf-}\Delta'' \text{ source-out' nontrivial' eq}]$
obtain f **where** $f: \text{flow } \Delta'' f$
and $\text{value-flow } \Delta'' f = \alpha$
and $\text{IN}: \bigwedge x. d\text{-IN } f x \leq \text{value-flow } \Delta'' f$ **unfolding** *eq real* **using** *nn* **by** *blast*
hence $\text{flow } \Delta (\text{collect } f)$ **and** $\text{value-flow } \Delta (\text{collect } f) = \alpha$ **by**(*simp-all add*: *value-collect*)
moreover {
fix x
have $d\text{-IN } (\text{collect } f) x = (\sum^+_{y \in \text{range } (\lambda y. \text{Edge } y x)}. f (y, \text{Vertex } x))$
by(*simp add*: *nn-integral-count-space-reindex d-IN-def*)
also **have** $\dots \leq d\text{-IN } f (\text{Vertex } x)$ **unfolding** *d-IN-def*
by (*auto intro*!: *nn-integral-mono simp add*: *nn-integral-count-space-indicator split: split-indicator*)
also **have** $\dots \leq \text{value-flow } \Delta (\text{collect } f)$ **using** $\text{IN}[\text{of Vertex } x] f$ **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:  $x \notin \text{range Inner} \longleftrightarrow x = \text{SOURCE}$ 
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  $\Rightarrow$  'v node  $\Rightarrow$  bool
where
  SOURCE: edge' SOURCE (Inner (source  $\Delta$ ))
| Inner: edge  $\Delta$  x y  $\Longrightarrow$  edge' (Inner x) (Inner y)

```

```

private inductive-simps edge'-simps [simp]:
  edge' SOURCE x
  edge' (Inner y) x
  edge' y SOURCE
  edge' y (Inner x)

```

```

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

```

```

private lemma capacity'-source-in [simp]: capacity' (y, Inner (source  $\Delta$ )) = (if y = SOURCE then  $\alpha$  else 0)
by(cases y)(simp-all add: capacity-outside source-in)

```

```

private definition  $\Delta'$  :: 'v node network
where  $\Delta' = (\text{edge} = \text{edge}', \text{capacity} = \text{capacity}', \text{source} = \text{SOURCE}, \text{sink} = \text{Inner} (\text{sink } \Delta))$ 

```

```

private lemma  $\Delta'$ -sel [simp]:
  edge  $\Delta' = \text{edge}'$ 
  capacity  $\Delta' = \text{capacity}'$ 
  source  $\Delta' = \text{SOURCE}$ 
  sink  $\Delta' = \text{Inner} (\text{sink } \Delta)$ 
by(simp-all add:  $\Delta'$ -def)

```

```

private lemma E- $\Delta'$ :  $\mathbf{E}_{\Delta'} = \{(\text{SOURCE}, \text{Inner} (\text{source } \Delta))\} \cup (\lambda(x, y). (\text{Inner } x, \text{Inner } y)) \text{ ` } \mathbf{E}$ 

```

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** $\mathbf{E}-\Delta'$ **by** *simp*

show source $\Delta' \neq$ sink Δ' **by** *simp*

show capacity $\Delta' e = 0$ **if** $e \notin \mathbf{E}_{\Delta'}$ **for** e **using** *that* **unfolding** $\mathbf{E}-\Delta'$

by(*cases e rule: capacity'.cases*)(*auto intro: capacity-outside*)

show capacity $\Delta' e \neq \top$ **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 $d\text{-IN}(\text{capacity } \Delta') x \neq \top \vee d\text{-OUT}(\text{capacity } \Delta') x \neq \top$ **if** sink: $x \neq$ sink Δ' **for** x

proof(*cases x*)

case (*Inner x'*)

consider (*source*) $x' = \text{source } \Delta \mid (IN) x' \neq \text{source } \Delta$ $d\text{-IN}(\text{capacity } \Delta) x' \neq \top \mid (OUT) d\text{-OUT}(\text{capacity } \Delta) x' \neq \top$

using *finite-capacity[of x'] sink Inner* **by**(*auto*)

thus *?thesis*

proof(*cases*)

case *source*

with *Inner* **have** $d\text{-IN}(\text{capacity } \Delta') x = (\sum^+ y. \alpha * \text{indicator } \{SOURCE :: 'v \text{ node}\} y)$

unfolding *d-IN-def* **by**(*intro nn-integral-cong*)(*simp split: split-indicator*)

also **have** $\dots = \alpha$ **by**(*simp add: max-def*)

finally **show** *?thesis* **using** *assms* **by** *simp*

next

case *IN*

with *Inner* **have** $d\text{-IN}(\text{capacity } \Delta') x = (\sum^+ y \in \text{range } Inner. \text{capacity } \Delta (node.inner y, x'))$

by(*auto simp add: d-IN-def nn-integral-count-space-indicator not-Inner-conv intro!: nn-integral-cong split: split-indicator*)

also **have** $\dots = d\text{-IN}(\text{capacity } \Delta) x'$ **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** $d\text{-OUT}(\text{capacity } \Delta') x = (\sum^+ y \in \text{range } Inner. \text{capacity } \Delta (x', node.inner y))$

```

    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  $\Delta$ )  $x'$  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  $\neg$  edge  $\Delta' x$  for  $x$  by(cases  $x$ )(simp-all add: no-loop)
show  $\neg$  edge  $\Delta' x$  (source  $\Delta'$ ) for  $x$  by simp
qed
qed

```

```

private fun lift :: 'v flow  $\Rightarrow$  'v node flow

```

```

where

```

```

  lift f (SOURCE, Inner y) = (if y = source  $\Delta$  then value-flow  $\Delta$  f else 0)
| lift f (Inner x, Inner y) = f (x, y)
| lift f - = 0

```

```

private lemma d-OUT-lift-Inner [simp]: d-OUT (lift f) (Inner x) = d-OUT f x
(is ?lhs = ?rhs)

```

```

proof -

```

```

  have ?lhs = ( $\sum^+$   $y \in \text{range Inner}$ . 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
 $\Delta$  f (is ?lhs = ?rhs)

```

```

proof -

```

```

  have ?lhs = ( $\sum^+$   $y$ . lift f (SOURCE, y) * indicator {Inner (source  $\Delta$ )} 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  $x \neq \text{source } \Delta$ 
  shows d-IN (lift f) (Inner x) = d-IN f x (is ?lhs = ?rhs)

```

```

proof -

```

```

  have ?lhs = ( $\sum^+$   $y \in \text{range Inner}$ . 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  $\Delta$ )) = value-flow
 $\Delta$  f + d-IN f (source  $\Delta$ ) (is ?lhs = ?rhs)

```

```

proof –
  have ?lhs = ( $\sum^+ y. \text{lift } f (y, \text{Inner } (\text{source } \Delta)) * \text{indicator } \{SOURCE\} y$ ) +
  ( $\sum^+ y \in \text{range } \text{Inner}. \text{lift } f (y, \text{Inner } (\text{source } \Delta))$ )
  (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  $\Delta$ ) by(simp add: nn-integral-count-space-reindex
d-IN-def)
  also have ?SOURCE = value-flow  $\Delta$  f
  by(simp add: max-def one-ennreal-def[symmetric] )
  finally show ?thesis .
qed

private lemma flow-lift [simp]:
  assumes flow  $\Delta$  f
  shows flow  $\Delta'$  (lift f)
proof
  show lift f e  $\leq$  capacity  $\Delta'$  e for e
  by(cases e rule: capacity'.cases)(auto intro: flowD-capacity[OF assms] simp add:
 $\alpha$ -def intro: SUP-upper2 assms)

  fix x
  assume x: x  $\neq$  source  $\Delta'$  x  $\neq$  sink  $\Delta'$ 
  then obtain x' where x': x = Inner x' by(cases x) auto
  then show KIR (lift f) x using x
  by(cases x' = source  $\Delta$ )(auto simp add: flowD-source-IN[OF assms] dest:
flowD-KIR[OF assms])
qed

private abbreviation (input) unlift :: 'v node flow  $\Rightarrow$  'v flow
where unlift f  $\equiv$  ( $\lambda(x, y). f (\text{Inner } x, \text{Inner } y)$ )

private lemma flow-unlift [simp]:
  assumes f: flow  $\Delta'$  f
  shows flow  $\Delta$  (unlift f)
proof
  show unlift f e  $\leq$  capacity  $\Delta$  e for e using flowD-capacity[OF f, of map-prod
Inner Inner e]
  by(cases e)(simp)
next
  fix x
  assume x: x  $\neq$  source  $\Delta$  x  $\neq$  sink  $\Delta$ 
  have d-OUT (unlift f) x = ( $\sum^+ y \in \text{range } \text{Inner}. f (\text{Inner } x, y)$ )
  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 $\dots = d-IN\ f\ (Inner\ x)$ **using** $x\ flowD-KIR[OF\ f,\ of\ Inner\ x]$ **by**(*simp*)

also have $\dots = (\sum^+ y \in range\ Inner.\ f\ (y,\ Inner\ x))$

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 $\dots = d-IN\ (unlift\ f)\ x$ **by**(*simp add: nn-integral-count-space-reindex d-IN-def*)

finally show $KIR\ (unlift\ f)\ x$.

qed

private lemma *value-unlift*:

assumes $f: flow\ \Delta'\ f$

shows $value-flow\ \Delta\ (unlift\ f) = value-flow\ \Delta'\ f$

proof –

have $value-flow\ \Delta\ (unlift\ f) = (\sum^+ y \in range\ Inner.\ f\ (Inner\ (source\ \Delta),\ y))$

by(*simp add: nn-integral-count-space-reindex d-OUT-def*)

also have $\dots = d-OUT\ f\ (Inner\ (source\ \Delta))$ **using** $flowD-capacity[OF\ f,\ of\ (Inner\ (source\ \Delta),\ SOURCE)]$

by(*auto simp add: nn-integral-count-space-indicator d-OUT-def not-Inner-conv intro!: nn-integral-cong split: split-indicator*)

also have $\dots = d-IN\ f\ (Inner\ (source\ \Delta))$ **using** $flowD-KIR[OF\ f,\ of\ Inner\ (source\ \Delta)]$ **by**(*simp*)

also have $\dots = (\sum^+ y.\ f\ (y,\ Inner\ (source\ \Delta)) * indicator\ \{SOURCE\}\ y)$

unfolding $d-IN-def$ **using** $flowD-capacity[OF\ f,\ of\ (x,\ Inner\ (source\ \Delta))\ for\ x]$

by(*intro nn-integral-cong*)(*auto intro!: antisym split: split-indicator if-split-asm elim: meta-allE*)

also have $\dots = f\ (SOURCE,\ Inner\ (source\ \Delta))$ **by** *simp*

also have $\dots = (\sum^+ y.\ f\ (SOURCE,\ y) * indicator\ \{Inner\ (source\ \Delta)\}\ y)$

by(*simp add: one-ennreal-def[symmetric]*)

also have $\dots = value-flow\ \Delta'\ f$ **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*:

$\exists f.\ flow\ \Delta\ f \wedge value-flow\ \Delta\ f = \alpha \wedge (\forall x.\ d-IN\ f\ x \leq value-flow\ \Delta\ f)$

proof(*cases* α)

case (*real* α')

hence $\alpha: \alpha \neq \top$ **by** *simp*

then interpret Δ' : *flow-attainability* Δ' **by**(*rule* Δ' -*flow-attainability*)

```

have source-out: edge  $\Delta'$  (source  $\Delta'$ )  $y \longleftrightarrow y = \text{Inner}$  (source  $\Delta$ ) for  $y$  by (auto)
have nontrivial:  $\mathbf{E}_{\Delta'} \neq \{\}$  by (auto intro: edge'.intros)

have eq: ( $\text{SUP } g \in \{g. \text{flow } \Delta' g\}. \text{value-flow } \Delta' g$ ) = ( $\text{SUP } g \in \{g. \text{flow } \Delta g\}. \text{value-flow } \Delta g$ ) (is ?lhs = ?rhs)
proof (intro antisym SUP-least; unfold mem-Collect-eq)
  fix  $g$ 
  assume  $g: \text{flow } \Delta' g$ 
  hence  $\text{value-flow } \Delta' g = \text{value-flow } \Delta (\text{unlift } g)$  by (simp add: value-unlift)
  also { from  $g$  have  $\text{flow } \Delta (\text{unlift } g)$  by simp }
  then have ...  $\leq ?rhs$  by (blast intro: SUP-upper2)
  finally show  $\text{value-flow } \Delta' g \leq \dots$  .
next
  fix  $g$ 
  assume  $g: \text{flow } \Delta g$ 
  hence  $\text{value-flow } \Delta g = \text{value-flow } \Delta' (\text{lift } g)$  by simp
  also { from  $g$  have  $\text{flow } \Delta' (\text{lift } g)$  by simp }
  then have ...  $\leq ?lhs$  by (blast intro: SUP-upper2)
  finally show  $\text{value-flow } \Delta g \leq ?lhs$  .
qed
also have ... = ennreal  $\alpha'$  using real by (simp add:  $\alpha$ -def)
finally obtain  $f$  where  $f: \text{flow } \Delta' f$ 
  and  $\text{value-f: value-flow } \Delta' f = (\bigsqcup g \in \{g. \text{flow } \Delta' g\}. \text{value-flow } \Delta' g)$ 
  and  $\text{IN-f: } \bigwedge x. d\text{-IN } f x \leq \text{value-flow } \Delta' f$ 
  using  $\langle 0 \leq \alpha' \rangle$  by (blast dest:  $\Delta'.ex\text{-max-flow}''$  [OF source-out nontrivial])
have  $\text{flow } \Delta (\text{unlift } f)$  using  $f$  by simp
moreover have  $\text{value-flow } \Delta (\text{unlift } f) = \alpha$  using  $f$  eq value-f by (simp add:
value-unlift  $\alpha$ -def)
moreover {
  fix  $x$ 
  have  $d\text{-IN } (\text{unlift } f) x = (\sum^+ y \in \text{range Inner}. f(y, \text{Inner } x))$ 
  by (simp add: nn-integral-count-space-reindex d-IN-def)
  also have ...  $\leq d\text{-IN } f (\text{Inner } x)$  unfolding d-IN-def
  by (auto intro!: nn-integral-mono simp add: nn-integral-count-space-indicator
split: split-indicator)
  also have ...  $\leq \text{value-flow } \Delta (\text{unlift } f)$  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  $\exists f. \text{flow } \Delta f \wedge \text{value-flow } \Delta f = \top$ )
  case True
  with top show ?thesis by auto
next
case False
  hence real:  $\forall f. \alpha = \top \longrightarrow \text{flow } \Delta f \longrightarrow \text{value-flow } \Delta f < \alpha$  using top by

```



```

(auto simp: less-top)
{ fix i
  have  $2 * 2^i < \alpha$  using top by (simp-all add: ennreal-mult-less-top
power-less-top-ennreal)
  from flow-by-value[OF this real] have  $\exists f. \text{flow } \Delta f \wedge \text{value-flow } \Delta f = 2 * 2^i$  by blast }
  then obtain f-i where f-i:  $\bigwedge i. \text{flow } \Delta (f-i \ i)$ 
  and value-i:  $\bigwedge i. \text{value-flow } \Delta (f-i \ i) = 2 * 2^i$  by metis
  define f where  $f \ e = (\sum^+ i. f-i \ i \ e / (2 * 2^i))$  for e
  have flow  $\Delta f$ 
  proof
    fix e
    have  $f \ e \leq (\sum^+ i. (\text{SUP } i. f-i \ i \ e) / (2 * 2^i))$  unfolding f-def
    by(rule nn-integral-mono)(auto intro!: divide-right-mono-ennreal SUP-upper)
    also have  $\dots = (\text{SUP } i. f-i \ i \ e) / 2 * (\sum^+ i. 1 / 2^i)$ 
    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  $(\sum^+ i. 1 / 2^i) = (\sum i. \text{ennreal } ((1 / 2)^i))$ 
    by(simp add: nn-integral-count-space-nat power-divide divide-ennreal[symmetric]
ennreal-power[symmetric])
    also have  $\dots = \text{ennreal } (\sum i. (1 / 2)^i)$ 
    by(intro suminf-ennreal2 complete-algebra-summable-geometric) simp-all
    also have  $\dots = 2$  by(subst suminf-geometric; simp)
    also have  $(\text{SUP } i. f-i \ i \ e) / 2 * 2 = (\text{SUP } i. f-i \ i \ e)$ 
    by(simp add: ennreal-divide-times)
    also have  $\dots \leq \text{capacity } \Delta e$  by(rule SUP-least)(rule flowD-capacity[OF f-i])
    finally show  $f \ e \leq \text{capacity } \Delta e$  .

  fix x
  assume x:  $x \neq \text{source } \Delta x \neq \text{sink } \Delta$ 
  have d-OUT  $f \ x = (\sum^+ i \in \text{UNIV}. \sum^+ y. f-i \ i \ (x, y) / (2 * 2^i))$ 
  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  $\dots = (\sum^+ i. d-OUT \ (f-i \ i) \ x / (2 * 2^i))$  unfolding d-OUT-def
  by(simp add: nn-integral-divide)
  also have  $\dots = (\sum^+ i. d-IN \ (f-i \ i) \ x / (2 * 2^i))$  by(simp add:
flowD-KIR[OF f-i, OF x])
  also have  $\dots = (\sum^+ i \in \text{UNIV}. \sum^+ y. f-i \ i \ (y, x) / (2 * 2^i))$ 
  by(simp add: nn-integral-divide d-IN-def)
  also have  $\dots = d-IN \ f \ x$  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  $\Delta f = (\sum^+ i. \text{value-flow } \Delta (f-i \ i) / (2 * 2^i))$ 

```

```

    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 ... =  $\top$ 
    by(simp add: value-i ennreal-mult-less-top power-less-top-ennreal)
    finally have value-flow  $\Delta f = \top$  .
  }
  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*: $SINK (plus-current f g) = SINK f \cap SINK g$
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* :: $('v, 'more) \text{ web-scheme} \Rightarrow 'v \text{ current} \Rightarrow 'v \text{ current} \Rightarrow 'v \text{ current}$ ($\hookleftarrow \curvearrowright_1 \rightarrow$ [66, 66] 65)

where $plus-web \Gamma f g \equiv plus-current f (g \upharpoonright \Gamma / f)$

lemma *d-OUT-plus-web*:

fixes Γ (**structure**)

shows $d-OUT (f \curvearrowright g) x = d-OUT f x + d-OUT (g \upharpoonright \Gamma / f) x$ (**is** ?lhs = ?rhs)

proof –

have ?lhs = $d-OUT f x + (\sum^+ y. (if x \in RF^\circ (TER f) \text{ then } 0 \text{ else } g(x, y) * indicator (- RF (TER f)) y))$

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* = $d-IN f y + (\sum^+ x. (if\ y \in RF\ (TER\ f)\ then\ 0\ else\ g\ (x, y) * indicator\ (-\ RF^\circ\ (TER\ f))\ x))$

unfolding *d-IN-def* **by**(*subst nn-integral-add[symmetric]*)(*auto intro!: nn-integral-cong split: split-indicator*)

also have $\dots = ?rhs$ **by**(*auto simp add: d-IN-def intro!: arg-cong2[where f=(+)] nn-integral-cong*)

finally show *?thesis* .

qed

lemma *plus-web-greater*: $f\ e \leq (f \frown_\Gamma g)\ e$

by(*cases e*)(*auto split: split-indicator*)

lemma *current-plus-web*:

fixes Γ (**structure**)

shows $\llbracket current\ \Gamma\ f; wave\ \Gamma\ f; current\ \Gamma\ g \rrbracket \implies current\ \Gamma\ (f \frown g)$

by(*blast intro: current-plus-current current-restrict-current*)

context

fixes $\Gamma :: ('v, 'more)\ web-scheme$ (**structure**)

and $f\ g :: 'v\ current$

assumes $f: current\ \Gamma\ f$

and $w: wave\ \Gamma\ f$

and $g: current\ \Gamma\ g$

begin

context

fixes $x :: 'v$

assumes $x: x \in \mathcal{E}\ (TER\ f \cup TER\ g)$

begin

qualified lemma *RF-f*: $x \notin RF^\circ\ (TER\ f)$

proof

assume $*$: $x \in RF^\circ\ (TER\ f)$

from x **obtain** $p\ y$ **where** $p: path\ \Gamma\ x\ p\ y$ **and** $y: y \in B\ \Gamma$

and *bypass*: $\bigwedge z. \llbracket x \neq y; z \in set\ p \rrbracket \implies z = x \vee z \notin TER\ f \cup TER\ g$ **by**(*rule $\mathcal{E}$ -E*) *blast*

from *rtrancl-path-distinct*[*OF p*] **obtain** p'

where $p: path\ \Gamma\ x\ p'\ y$ **and** $p': set\ p' \subseteq set\ p$ **and** *distinct*: $distinct\ (x \# p')$.

from $*$ **have** $x': x \in RF\ (TER\ f)$ **and** $\mathcal{E}: x \notin \mathcal{E}\ (TER\ f)$ **by**(*auto simp add:*

roofed-circ-def)

hence $x \notin \text{TER } f$ using *not-essentialD*[$\text{OF} - p \ y$] p' bypass by *blast*
 with *roofedD*[$\text{OF} \ x' \ p \ y$] obtain z where $z: z \in \text{set } p' \ z \in \text{TER } f$ by *auto*
 with p have $y \in \text{set } p'$ 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*: $x \notin \text{RF}^\circ (\text{TER } g)$

proof

assume *: $x \in \text{RF}^\circ (\text{TER } g)$

from x obtain $p \ y$ where $p: \text{path } \Gamma \ x \ p \ y$ and $y: y \in B \ \Gamma$
 and *bypass*: $\bigwedge z. \llbracket x \neq y; z \in \text{set } p \rrbracket \implies z = x \vee z \notin \text{TER } f \cup \text{TER } g$ by(*rule*
 $\mathcal{E}\text{-E}$) *blast*

from *rtrancl-path-distinct*[$\text{OF } p$] obtain p'
 where $p: \text{path } \Gamma \ x \ p' \ y$ and $p': \text{set } p' \subseteq \text{set } p$ and *distinct*: *distinct* $(x \# p')$.

from * have $x': x \in \text{RF} (\text{TER } g)$ and $\mathcal{E}: x \notin \mathcal{E} (\text{TER } g)$ by(*auto simp add:*
roofed-circ-def)

hence $x \notin \text{TER } g$ using *not-essentialD*[$\text{OF} - p \ y$] p' bypass by *blast*
 with *roofedD*[$\text{OF} \ x' \ p \ y$] obtain z where $z: z \in \text{set } p' \ z \in \text{TER } g$ by *auto*
 with p have $y \in \text{set } p'$ 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*: $x \in \text{SINK} (g \upharpoonright \Gamma / f)$ (*is* $- \in \text{SINK} ?g$)

shows $x \in \text{TER} (f \frown g)$

proof

from x obtain $p \ y$ where $p: \text{path } \Gamma \ x \ p \ y$ and $y: y \in B \ \Gamma$
 and *bypass*: $\bigwedge z. \llbracket x \neq y; z \in \text{set } p \rrbracket \implies z = x \vee z \notin \text{TER } f \cup \text{TER } g$ by(*rule*
 $\mathcal{E}\text{-E}$) *blast*

from *rtrancl-path-distinct*[$\text{OF } p$] obtain p'
 where $p: \text{path } \Gamma \ x \ p' \ y$ and $p': \text{set } p' \subseteq \text{set } p$ and *distinct*: *distinct* $(x \# p')$.

from *RF-f* have $x \in \text{SINK } f$

by(*auto simp add: roofed-circ-def SINK.simps dest: waveD-OUT*[$\text{OF } w$])

thus $x \in \text{SINK} (f \frown g)$ using *SINK*

by(*simp add: SINK.simps d-OUT-plus-web*)

show $x \in \text{SAT } \Gamma (f \frown g)$

proof(*cases* $x \in \text{TER } f$)

case *True*

hence $x \in \text{SAT } \Gamma \ f$ by *simp*

moreover have $\dots \subseteq \text{SAT } \Gamma (f \frown g)$ by(*rule SAT-mono plus-web-greater*)+

ultimately show *?thesis* by *blast*

next

case *False*

```

with x have x ∈ TER g by auto
from False RF-f have x ∉ RF (TER f) by(auto simp add: roofed-circ-def)
moreover { fix z
  assume z: z ∈ RFo (TER f)
  have (z, x) ∉ E
  proof
    assume (z, x) ∈ E
    hence path': path Γ z (x # p') y using p by(simp add: rtranc-path.step)
    from z have z ∈ RF (TER f) by(simp add: roofed-circ-def)
    from roofedD[OF this path' y] False
    consider (path) z' where z' ∈ set p' z' ∈ TER f | (TER) z ∈ TER f by
auto
  then show False
  proof cases
    { case (path z')
      with p distinct have x ≠ y
      by(auto 4 3 intro: last-in-set elim: rtranc-path.cases dest: rtranc-path-last[symmetric])
      from bypass[OF this, of z'] path False p' show False by auto }
    note that = this
    case TER
    with z have ¬ 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' ∈ set p' z'
    ∈ 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 ∩ g) x ≥ d-IN g x
by(simp add: d-IN-plus-web)
with ⟨x ∈ TER g⟩ show ?thesis by(auto elim!: SAT.cases intro: SAT.intros)
qed
qed

qualified lemma SINK-TER-in'':
  assumes ∧x. x ∉ RF (TER g) ⇒ d-OUT g x = 0
  shows x ∈ 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 ∩ Γ / f) ⇒ wave Γ (f ∩ g)
using f w by(rule wave-plus-current)(rule current-restrict-current[OF w g])

lemma TER-plus-web'':
  assumes ∧x. x ∉ RF (TER g) ⇒ d-OUT g x = 0
  shows E (TER f ∪ TER g) ⊆ TER (f ∩ 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':  $wave\ \Gamma\ g \implies \mathcal{E} (TER\ f \cup TER\ g) \subseteq TER\ (f \frown g)$ 
by(rule TER-plus-web'')(rule waveD-OUT)

lemma wave-plus':  $wave\ \Gamma\ g \implies wave\ \Gamma\ (f \frown g)$ 
by(rule wave-plus)(rule wave-restrict-current[OF f w g])

end

lemma RF-TER-plus-web:
  fixes  $\Gamma$  (structure)
  assumes  $f$ : current  $\Gamma\ f$ 
  and  $w$ : wave  $\Gamma\ f$ 
  and  $g$ : current  $\Gamma\ g$ 
  and  $w'$ : wave  $\Gamma\ g$ 
  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  $\dots \subseteq RF\ (TER\ (f \frown g))$  .
next
  have  $fg$ : current  $\Gamma\ (f \frown g)$  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  $\Gamma\ x\ p\ y$  and  $y$ :  $y \in B\ \Gamma$ 
    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  $z \in SAT\ \Gamma\ (f \frown g)$  by simp
    hence  $SAT$ :  $z \in SAT\ \Gamma\ f \cup SAT\ \Gamma\ g$ 
    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  $(\exists z \in set\ p. z \in TER\ f \cup TER\ g) \vee x \in TER\ f \cup TER\ g$ 
  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  $\Gamma$  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  $\Gamma$  (structure)
  assumes f:  $\bigwedge f. f \in Y \implies \text{current } \Gamma f$ 
  and w:  $\bigwedge f. f \in Y \implies \text{wave } \Gamma f$ 
  and Y: Complete-Partial-Order.chain ( $\leq$ ) Y  $Y \neq \{\}$  countable (support-flow
(Sup Y))
  shows RF (TER (Sup Y)) = RF ( $\bigcup_{f \in Y}. \text{TER } f$ )
proof (rule set-eqI iffI) +
  fix x
  assume x:  $x \in \text{RF } (\text{TER } (\text{Sup } Y))$ 
  have x  $\in \text{RF } (\text{RF } (\bigcup_{f \in Y}. \text{TER } f))$ 
  proof
    fix p y
    assume p: path  $\Gamma$  x p y and y:  $y \in B \Gamma$ 
    from roofedD[OF x p y] obtain z where z:  $z \in \text{set } (x \# p)$  and TER:  $z \in$ 
TER (Sup Y) by auto
    from TER have SINK:  $z \in \text{SINK } f \text{ if } f \in Y \text{ for } f$  using that by (auto simp
add: SINK-Sup[OF Y])

    from Y(2) obtain f where y:  $f \in Y$  by blast

    show  $(\exists z \in \text{set } p. z \in \text{RF } (\bigcup_{f \in Y}. \text{TER } f)) \vee x \in \text{RF } (\bigcup_{f \in Y}. \text{TER } f)$ 
  proof (cases  $\exists f \in Y. z \in \text{RF } (\text{TER } f)$ )
    case True
    then obtain f where fY:  $f \in Y$  and zf:  $z \in \text{RF } (\text{TER } f)$  by blast
    from zf have z  $\in \text{RF } (\bigcup_{f \in Y}. \text{TER } f)$  by (rule in-roofed-mono)(auto intro:
fY)
    with z show ?thesis by auto
  next
    case False
    hence *:  $d\text{-IN } f z = 0 \text{ if } f \in Y \text{ for } f$  using that by (auto intro: wave-not-RF-IN-zero[OF
f w])
    hence  $d\text{-IN } (\text{Sup } Y) z = 0$  using Y(2) by (simp add: d-IN-Sup[OF Y])
    with TER have z  $\in \text{SAT } \Gamma f$  using *[OF y]
    by (simp add: SAT.simps)
    with SINK[OF y] have z  $\in \text{TER } f$  by simp
    with z y show ?thesis by (auto intro: roofed-greaterI)
  qed
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  $p: path \Gamma x p y$  and  $y: y \in B \Gamma$ 
  from roofedD[OF  $x p y$ ] obtain  $z f$  where  $z \in set (x \# p)$ 
    and  $z: f \in Y$  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  $\Gamma (Sup Y)$  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  $z \in SAT \Gamma (Sup Y)$  using TER SAT-Sup-upper[OF  $z$ , of
 $\Gamma$ ] 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  $\varepsilon$ 
  defines  $\varepsilon \equiv (\int^+ y. u y \partial count-space (B \Gamma))$ 
  assumes u-outside:  $\bigwedge x. x \notin B \Gamma \implies u x = 0$ 
  and finite-ε:  $\varepsilon \neq \top$ 
begin

```

```

private lemma u-A:  $x \in A \Gamma \implies u x = 0$ 
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  $\varepsilon$ -def by(rule nn-integral-ge-point)(simp add: True)
    also have  $\dots < \top$  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*: $u \leq \text{weight } \Gamma$
assumes *hindered-by*: *hindered-by* ($\Gamma \llbracket \text{weight} := \text{weight } \Gamma - u \rrbracket$) ε (**is** *hindered-by* $\text{?}\Gamma$ -)
shows *hindered* Γ
proof —
note [*simp*] = *u-finite*
let *?TER* = *TER* $\text{?}\Gamma$
from *hindered-by* **obtain** *f*
where *hindrance-by*: *hindrance-by* $\text{?}\Gamma$ *f* ε
and *f*: *current* $\text{?}\Gamma$ *f*
and *w*: *wave* $\text{?}\Gamma$ *f* **by cases**
from *hindrance-by* **obtain** *a* **where** *a*: $a \in A \ \Gamma \ a \notin \mathcal{E}_{\text{?}\Gamma} \ (\text{?TER } f)$
and *a-le*: *d-OUT* *f* *a* $< \text{weight } \Gamma \ a$
and *ε-less*: $\text{weight } \Gamma \ a - \text{d-OUT } f \ a > \varepsilon$
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* ($\langle \cdot \rangle$)

define *edge'*

where *edge'* *xo yo* =
(*case* (*xo, yo*) *of*
(*None, Some y*) $\Rightarrow y \in \mathbf{V} \wedge y \notin A \ \Gamma$
| (*Some x, Some y*) $\Rightarrow \text{edge } \Gamma \ x \ y \vee \text{edge } \Gamma \ y \ x$
| - $\Rightarrow \text{False}$) **for** *xo yo*

define *cap*

where *cap* *e* =
(*case* *e* *of*
(*None, Some y*) $\Rightarrow$ *if* $y \in \mathbf{V}$ *then* *u* *else* 0
| (*Some x, Some y*) $\Rightarrow$ *if* $\text{edge } \Gamma \ x \ y \wedge x \neq a$ *then* *f* (*x, y*) *else if* $\text{edge } \Gamma \ y \ x$
then $\max (\text{weight } \Gamma \ x) (\text{weight } \Gamma \ y) + 1$ *else* 0
| - $\Rightarrow 0$) **for** *e*

define Ψ **where** $\Psi = \llbracket \text{edge} = \text{edge}', \text{capacity} = \text{cap}, \text{source} = \text{None}, \text{sink} = \text{Some } a \rrbracket$

have *edge'-simps* [*simp*]:

edge' None $\langle y \rangle \longleftrightarrow y \in \mathbf{V} \wedge y \notin A \ \Gamma$
edge' xo None $\longleftrightarrow \text{False}$
edge' $\langle x \rangle \langle y \rangle \longleftrightarrow \text{edge } \Gamma \ x \ y \vee \text{edge } \Gamma \ y \ x$
for *xo x y* **by**(*simp-all add: edge'-def split: option.split*)

have *edge-None1E* [*elim!*]: *thesis if* *edge' None y* $\wedge z. \llbracket y = \langle z \rangle; z \in \mathbf{V}; z \notin A \ \Gamma \rrbracket \Rightarrow$ *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 \wedge z. \llbracket y = \langle z \rangle; \text{edge } \Gamma x z \vee \text{edge } \Gamma z x \rrbracket \implies \text{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 \llbracket x = \text{None}; y \in \mathbf{V}; y \notin A \Gamma \rrbracket \implies \text{thesis} \wedge z. \llbracket x = \langle z \rangle; \text{edge } \Gamma z y \vee \text{edge } \Gamma y z \rrbracket \implies \text{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 0 else 0)
  cap ( $x_0$ , None) = 0
  cap ( $\langle x \rangle$ ,  $\langle y \rangle$ ) =
    (if edge  $\Gamma x y \wedge x \neq a$  then f ( $x, y$ ) else if edge  $\Gamma y x$  then max (weight  $\Gamma x$ )
    (weight  $\Gamma y$ ) + 1 else 0)
  for  $x_0 x y$  by(simp-all add: cap-def split: option.split)

have  $\Psi$ -sel [simp]:
  edge  $\Psi$  = edge'
  capacity  $\Psi$  = cap
  source  $\Psi$  = None
  sink  $\Psi$  =  $\langle a \rangle$ 
  by(simp-all add:  $\Psi$ -def)

have cap-outside1:  $\neg \text{vertex } \Gamma x \implies \text{cap } (\langle x \rangle, y) = 0$  for  $x y$ 
  by(cases  $y$ )(auto simp add: vertex-def)
have capacity-A-weight: d-OUT cap  $\langle x \rangle \leq \text{weight } \Gamma x$  if  $x \in A \Gamma$  for  $x$ 
proof -
  have d-OUT cap  $\langle x \rangle \leq (\sum^+_{y \in \text{range } \text{Some}. f(x, y)})$ 
  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 ...  $\leq \text{weight } \Gamma x$  using f' by(rule currentD-weight-OUT)
  finally show ?thesis .
qed
have flow-attainability: flow-attainability  $\Psi$ 
proof
  have  $\mathbf{E}_\Psi = (\lambda(x, y). (\langle x \rangle, \langle y \rangle)) \text{ ' } \mathbf{E} \cup (\lambda(x, y). (\langle y \rangle, \langle x \rangle)) \text{ ' } \mathbf{E} \cup (\lambda x. (\text{None}, \langle x \rangle)) \text{ ' } (\mathbf{V} \cap - A \Gamma)$ 
  by(auto simp add: edge'-def split: option.split-asm)
  thus countable  $\mathbf{E}_\Psi$  by simp
next
  fix  $v$ 
  assume  $v \neq \text{sink } \Psi$ 
  consider (sink)  $v = \text{None} \mid (A) x$  where  $v = \langle x \rangle x \in A \Gamma$ 
   $\mid (B) y$  where  $v = \langle y \rangle y \notin A \Gamma y \in \mathbf{V} \mid (\text{outside}) x$  where  $v = \langle x \rangle x \notin \mathbf{V}$ 
  by(cases  $v$ ) auto
  then show d-IN (capacity  $\Psi$ )  $v \neq \top \vee$  d-OUT (capacity  $\Psi$ )  $v \neq \top$ 
  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  $\Psi$ )  $v \leq (\sum^+ x. f \text{ (the } x, y) * \text{indicator (range Some) } x$ 
+  $u \ y * \text{indicator \{None\} } x)$ 
    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  $\dots = (\sum^+ x \in \text{range Some}. f \text{ (the } x, y)) + u \ y$ 
  by(subst nn-integral-add)(simp-all add: nn-integral-count-space-indicator)
  also have  $\dots = d-IN \ f \ y + u \ y$  by(simp add: d-IN-def nn-integral-count-space-reindex)
  also have  $d-IN \ f \ y \leq \text{weight } \Gamma \ y$  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  $\Psi$ )  $v = 0$ 
  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  $\Psi \ e \neq \top$  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  $\Psi \ e = 0$  if  $e \notin \mathbf{E}_\Psi$  for e
  using that bipartite-V disjoint
  by(auto simp add: cap-def max-def intro: u-outside split: option.split prod.split)
  show  $\neg \text{edge } \Psi \ x \text{ (source } \Psi)$  for x by simp
  show  $\neg \text{edge } \Psi \ x \ x$  for x by(cases x)(simp-all add: no-loop)
  show source  $\Psi \neq \text{sink } \Psi$  by simp
qed
then interpret  $\Psi$ : flow-attainability  $\Psi$  .

define  $\alpha$  where  $\alpha = (\bigsqcup g \in \{g. \text{flow } \Psi \ g\}. \text{value-flow } \Psi \ g)$ 
have  $\alpha\text{-le}: \alpha \leq \varepsilon$ 
proof -
  have  $\alpha \leq d-OUT \ \text{cap None}$  unfolding  $\alpha$ -def by(rule SUP-least)(auto intro!:
d-OUT-mono dest: flowD-capacity)
  also have  $\dots \leq \int^+ y. \text{cap (None, } y) \ \partial \text{count-space (range Some)}$  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  $\dots \leq \varepsilon$  unfolding  $\varepsilon$ -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- $\varepsilon$  by (auto simp: top-unique)

from  $\Psi$ .ex-max-flow
obtain j where j: flow  $\Psi$  j
  and value-j: value-flow  $\Psi$  j =  $\alpha$ 
  and IN-j:  $\bigwedge x. d\text{-IN } j \ x \leq \alpha$ 
  unfolding  $\alpha$ -def by auto

have j-le-f:  $j \ (Some \ x, \ Some \ y) \leq f \ (x, \ y)$  if edge  $\Gamma \ x \ y$  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]:  $d\text{-IN } j \ x \neq \top$  for  $x$  using finite-flow by(rule neq-top-trans)(simp
add: IN-j)

have j-finite[simp]:  $j \ (x, \ y) < \top$  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:  $d\text{-OUT } j \ x \neq \top$  for  $x$ 
proof(cases  $x = source \ \Psi \vee x = sink \ \Psi$ )
  case True thus ?thesis
  proof cases
    case left thus ?thesis using finite-flow value-j by simp
  next
    case right
    have d-OUT (capacity  $\Psi$ )  $\langle a \rangle \neq \top$  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:  $d\text{-IN } j \ \langle x \rangle \leq weight \ \Gamma \ x$  for  $x$ 
proof(cases  $x \in A \ \Gamma$ )
  case xA: True
  show ?thesis
  proof(cases  $x = a$ )
    case True
    have d-IN  $j \ \langle x \rangle \leq \alpha$  by(rule IN-j)
    also have  $\dots \leq \varepsilon$  by(rule  $\alpha$ -le)
    also have  $\varepsilon < weight \ \Gamma \ a$  using  $\varepsilon$ -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  $d\text{-IN } j \langle x \rangle = d\text{-OUT } j \langle x \rangle$  using  $\text{flowD-KIR}[OF j, \text{ of Some } x]$  False by
simp
    also have  $\dots \leq d\text{-OUT cap } \langle x \rangle$  using  $\text{flowD-capacity}[OF j]$  by (auto intro:
d-OUT-mono)
    also have  $\dots \leq \text{weight } \Gamma x$  using  $xA$  by (rule capacity-A-weight)
    finally show ?thesis .
qed
next
case  $xA: \text{False}$ 
show ?thesis
proof (cases  $x \in B \Gamma$ )
case True
    have  $d\text{-IN } j \langle x \rangle \leq d\text{-IN cap } \langle x \rangle$  using  $\text{flowD-capacity}[OF j]$  by (auto intro:
d-IN-mono)
    also have  $\dots \leq (\sum^+ z. f (\text{the } z, x) * \text{indicator } (\text{range Some}) z) + (\sum^+ z
:: 'v \text{ option. } u x * \text{indicator } \{None\} z)$ 
    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  $\dots = d\text{-IN } f x + u x$ 
    by (simp add: nn-integral-count-space-indicator[symmetric] nn-integral-count-space-reindex
d-IN-def)
    also have  $\dots \leq \text{weight } \Gamma x$  using  $\text{currentD-weight-IN}[OF f, \text{ of } x]$  u-finite[of
x]
    using  $\varepsilon\text{-less } u$  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  $d\text{-IN } j \langle x \rangle = 0$  using False
    by (auto simp add: d-IN-def nn-integral-0-iff emeasure-count-space-eq-0 ver-
tex-def edge'-def split: option.split-asm intro!:  $\Psi.\text{flowD-outside}[OF j]$ )
    then show ?thesis
    by simp
qed
qed

let  $?j = j \circ \text{map-prod Some Some} \circ \text{prod.swap}$ 

have  $\text{finite-}j\text{-OUT}: (\sum^+ y \in \text{OUT } x. j (\langle x \rangle, \langle y \rangle)) \neq \top$  (is  $?j\text{-OUT} \neq -$ ) if  $x \in
A \Gamma$  for  $x$ 
    using  $\text{currentD-finite-OUT}[OF f', \text{ of } x]$ 
    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  $j\text{-OUT-eq}: ?j\text{-OUT } x = d\text{-OUT } j \langle x \rangle$  if  $x \in A \Gamma$  for  $x$ 
    proof -
        have  $?j\text{-OUT } x = (\sum^+ y \in \text{range Some. } j (\text{Some } x, y))$  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  $\Psi.\text{flowD-outside}[OF j]$  dest: bipartite-E split:

```

split-indicator)
also have $\dots = d\text{-OUT } j \langle x \rangle$
by(*auto simp add: d-OUT-def nn-integral-count-space-indicator notin-range-Some*
intro!: nn-integral-cong $\Psi.\text{flowD-outside}[OF j]$ 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*: $d\text{-OUT } g \ x = d\text{-OUT } f \ x + d\text{-IN } j \langle x \rangle - d\text{-OUT } j \langle x \rangle$ **if** $x \in A$
 Γ **for** x
proof –
have $d\text{-OUT } g \ x = (\sum^+ y \in \mathbf{OUT} \ x. f(x, y) + j(\langle y \rangle, \langle x \rangle) - j(\langle x \rangle, \langle y \rangle))$
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 $\dots = (\sum^+ y \in \mathbf{OUT} \ x. f(x, y) + j(\langle y \rangle, \langle x \rangle)) - (\sum^+ y \in \mathbf{OUT} \ x. j(\langle x \rangle, \langle y \rangle))$
(is $- = - - ?j\text{-OUT}$ **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 $\dots = (\sum^+ y \in \mathbf{OUT} \ x. f(x, y)) + (\sum^+ y \in \mathbf{OUT} \ x. j(\text{Some } y, \text{Some } x)) - ?j\text{-OUT}$
(is $- = ?f + ?j\text{-IN} - -$ **by**(*subst nn-integral-add*) *simp-all*
also have $?f = d\text{-OUT } f \ x$ **by**(*subst d-OUT-alt-def[where $G=\Gamma$]*)(*simp-all add: currentD-outside[OF f]*)
also have $?j\text{-OUT} = d\text{-OUT } j \langle x \rangle$ **using** *that* **by**(*rule j-OUT-eq*)
also have $?j\text{-IN} = (\sum^+ y \in \text{range } \text{Some}. j(y, \langle x \rangle))$ **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 $\Psi.\text{flowD-outside}[OF j]$ split: split-indicator dest: bipartite-E*)
also have $\dots = d\text{-IN } j(\text{Some } x)$ **using** *that disjoint*
by(*auto 4 3 simp add: d-IN-def nn-integral-count-space-indicator notin-range-Some intro!: nn-integral-cong $\Psi.\text{flowD-outside}[OF j]$ split: split-indicator*)
finally show *?thesis* **by** *simp*
qed

have *OUT-g-B*: $d\text{-OUT } g \ x = 0$ **if** $x \notin A$ Γ **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*: $d\text{-OUT } g \ a < \text{weight } \Gamma \ a$ **using** *a(1)*
proof –
have $d\text{-OUT } g \ a = d\text{-OUT } f \ a + d\text{-IN } j \langle a \rangle - d\text{-OUT } j \langle a \rangle$ **using** *a(1)* **by**(*rule OUT-g-A*)
also have $\dots \leq d\text{-OUT } f \ a + d\text{-IN } j \langle a \rangle$
by(*rule diff-le-self-ennreal*)
also have $\dots < \text{weight } \Gamma \ a + d\text{-IN } j \langle a \rangle - \varepsilon$

```

    using finite-ε ε-less currentD-finite-OUT[OF f']
    by (simp add: less-diff-eq-ennreal less-top ac-simps)
  also have ... ≤ 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 = (∑+ y∈range Some. j (y, ⟨x⟩)) by (simp add: d-OUT-def
nn-integral-count-space-reindex)
  also have ... = d-IN j ⟨x⟩ - (∑+ 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: (∑+ 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 ... = (∑+ 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 ... = (∑+ y∈IN x. f (y, x) + j (Some x, Some y)) - (∑+ 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 ... = (∑+ y∈IN x. f (y, x)) + (∑+ 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 = (∑+ y∈range Some. j (Some x, y)) using True disjoint
  by (simp add: nn-integral-count-space-reindex)(auto 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  $\Psi$ .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 = ( $\sum^+ y \in \text{range } \text{Some}. j (y, \text{Some } x)$ ) 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  $\Psi$ .flowD-outside[OF j] dest: bipartite-E split:
split-indicator)
    also have ... = d-IN j (Some x) - ( $\sum^+ y :: 'v \text{ option}. j (\text{None}, \text{Some } x) *$ 
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,  $\langle x \rangle$ ) = 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,  $\langle x \rangle$ )]
    by(cases vertex  $\Gamma$  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  $\Gamma$  g
proof
show d-OUT g x  $\leq$  weight  $\Gamma$  x for x
proof(cases x  $\in A$   $\Gamma$ )
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  $\leq$  weight  $\Gamma$  x for x
proof(cases x  $\in B$   $\Gamma$ )

```



```

    case False
    hence  $d\text{-IN } g \ x = 0$  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  $d\text{-IN } g \ x \leq (\text{weight } \Gamma \ x - u \ x) + u \ x$  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  $\dots = \text{weight } \Gamma \ x$ 
    using u by (intro diff-add-cancel-ennreal) (simp add: le-fun-def)
  finally show ?thesis .
qed
show  $g \ e = 0$  if  $e \notin \mathbf{E}$  for e using that
  by(cases e)(auto simp add: g-simps)
qed

define cap' where  $\text{cap}' = (\lambda(x, y). \text{if edge } \Gamma \ x \ y \text{ then } g \ (x, y) \text{ else if edge } \Gamma \ y \ x$ 
then 1 else 0)

have  $\text{cap}'\text{-simps}$  [simp]:  $\text{cap}' \ (x, y) = (\text{if edge } \Gamma \ x \ y \text{ then } g \ (x, y) \text{ else if edge } \Gamma$ 
 $y \ x \text{ then } 1 \text{ else } 0)$ 
  for x y by(simp add: cap'-def)

define G where  $G = (\lambda x \ y. \text{cap}' \ (x, y) > 0)$ 
have G-sel [simp]:  $\text{edge } G \ x \ y \longleftrightarrow \text{cap}' \ (x, y) > 0$  for x y by(simp add: G-def)
define reachable where  $\text{reachable } x = (\text{edge } G)^{**} \ x \ a$  for x
have reachable-alt-def:  $\text{reachable} \equiv \lambda x. \exists p. \text{path } G \ x \ p \ a$ 
  by(simp add: reachable-def [abs-def] rtranclp-eq-rtrancl-path)

have [simp]:  $\text{reachable } a$  by(auto simp add: reachable-def)

have AB-edge:  $\text{edge } G \ x \ y$  if  $\text{edge } \Gamma \ y \ x$  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:  $\text{reachable } y$  if  $\text{reachable } x \ (x, y) \in \mathbf{E}$  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  $\text{reachable } y \ (x, y) \in \mathbf{E} \neg \text{reachable } x$  for x y
proof(rule ccontr)
  assume  $g \ (x, y) \neq 0$ 
  then have  $g \ (x, y) > 0$  by (simp add: zero-less-iff-neq-zero)
  hence  $\text{edge } G \ x \ y$  using that by simp
  then have  $\text{reachable } x$  using reachable y
  unfolding reachable-def by(rule converse-rtranclp-into-rtranclp)
  with  $\langle \neg \text{reachable } x \rangle$  show False by contradiction
qed

```

```

have reachable-V: vertex  $\Gamma$   $x$  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:  $(\int^+ y. j \text{ (Some } y, \text{ Some } x) \partial \text{count-space } (\mathbf{IN} \ x)) \neq \top$  for  $x$ 
proof -
  have  $(\int^+ y. j \text{ (Some } y, \text{ Some } x) \partial \text{count-space } (\mathbf{IN} \ x)) \leq d\text{-IN } f \ x$ 
  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:  $j \text{ (} x, y \text{)} = 0$  if  $\neg \text{edge } \Psi \ x \ y$  for  $x \ y$ 
  using that flowD-capacity[OF  $j$ , of  $(x, y)$ ]  $\Psi$ .capacity-outside[of  $(x, y)$ ]
  by(auto)

define  $h$  where  $h = (\lambda(x, y). \text{ if reachable } x \wedge \text{ reachable } y \text{ then } g \text{ (} x, y \text{) else } 0)$ 
have h-simps [simp]:  $h \text{ (} x, y \text{)} = (\text{ if reachable } x \wedge \text{ reachable } y \text{ then } g \text{ (} x, y \text{) else } 0)$ 
0) for  $x \ y$ 
  by(simp add: h-def)
have h-le-g:  $h \ e \leq g \ e$  for  $e$  by(cases  $e$ ) simp

have OUT-h:  $d\text{-OUT } h \ x = (\text{ if reachable } x \text{ then } d\text{-OUT } g \ x \text{ else } 0)$  for  $x$ 
proof -
  have  $d\text{-OUT } h \ x = (\sum^+ y \in \mathbf{OUT} \ x. h \text{ (} x, y \text{)})$  using h-le-g currentD-outside[OF
 $g$ ]
  by(intro d-OUT-alt-def) auto
  also have  $\dots = (\sum^+ y \in \mathbf{OUT} \ x. \text{ if reachable } x \text{ then } g \text{ (} x, y \text{) else } 0)$ 
  by(auto intro!: nn-integral-cong simp add: outgoing-def dest: reachable-AB)
  also have  $\dots = (\text{ if reachable } x \text{ then } d\text{-OUT } g \ x \text{ else } 0)$ 
  by(auto intro!: d-OUT-alt-def[symmetric] currentD-outside[OF  $g$ ])
  finally show ?thesis .
qed
have IN-h:  $d\text{-IN } h \ x = (\text{ if reachable } x \text{ then } d\text{-IN } g \ x \text{ else } 0)$  for  $x$ 
proof -
  have  $d\text{-IN } h \ x = (\sum^+ y \in \mathbf{IN} \ x. h \text{ (} y, x \text{)})$ 
  using h-le-g currentD-outside[OF  $g$ ] by(intro d-IN-alt-def) auto
  also have  $\dots = (\sum^+ y \in \mathbf{IN} \ x. \text{ if reachable } x \text{ then } g \text{ (} y, x \text{) else } 0)$ 
  by(auto intro!: nn-integral-cong simp add: incoming-def dest: reachable-BA)
  also have  $\dots = (\text{ if reachable } x \text{ then } d\text{-IN } g \ x \text{ else } 0)$ 
  by(auto intro!: d-IN-alt-def[symmetric] currentD-outside[OF  $g$ ])
  finally show ?thesis .
qed

have  $h$ : current  $\Gamma$   $h$  using  $g$  h-le-g
proof(rule current-leI)

```

show $d\text{-OUT } h \ x \leq d\text{-IN } h \ x$ **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
 $\Psi.\text{flowD-outside}[OF j] \text{ zero-less-iff-neq-zero}$)
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** $q: \text{path } G \ y \ q \ a$ **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 $?E = \text{zip } (y \ \# \ q) \ q$
define E **where** $E = (None, \text{Some } y) \ \# \ \text{map } (\text{map-prod } \text{Some } \text{Some}) \ ?E$
define ζ **where** $\zeta = \text{Min } (\text{insert } (u \ y - j \ (None, \text{Some } y)) \ (\text{cap}' \text{ ' set } ?E))$
let $?j' = \lambda e. (\text{if } e \in \text{set } E \text{ then } \zeta \text{ else } 0) + j \ e$
define j' **where** $j' = \text{cleanup } ?j'$

have j-free: $0 < \text{cap}' \ e$ **if** $e \in \text{set } ?E$ **for** e **using** that **unfolding** $E\text{-def}$ list.sel
proof –
from that **obtain** i **where** $e: e = ((y \ \# \ q) ! i, q ! i)$
and $i: i < \text{length } q$ **by**(auto simp add: set-zip)
have $e': \text{edge } G \ ((y \ \# \ q) ! i) \ (q ! i)$ **using** $q \ i$ **by**(rule rtrancl-path-nth)
thus $?thesis$ **using** e **by**(simp)
qed

have $\zeta\text{-pos}: 0 < \zeta$ **unfolding** $\zeta\text{-def}$ **using** le
by(auto intro: j-free diff-gr0-ennreal)

have $\zeta\text{-le}: \zeta \leq \text{cap}' \ e$ **if** $e \in \text{set } ?E$ **for** e **using** that **unfolding** $\zeta\text{-def}$ **by** auto
have finite- ζ [simplified]: $\zeta < \top$ **unfolding** $\zeta\text{-def}$
by(intro Min-less-iff[THEN iffD2])(auto simp add: less-top[symmetric])

have $E\text{-antiparallel}: (x', y') \in \text{set } ?E \implies (y', x') \notin \text{set } ?E$ **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': $d\text{-OUT } ?j' \ x' = \zeta * \text{card } (\text{set } [(x'', y) \leftarrow E. x'' = x']) + d\text{-OUT}$
 $j \ x'$ **for** x'
proof –
have $d\text{-OUT } ?j' \ x' = d\text{-OUT } (\lambda e. (\text{if } e \in \text{set } E \text{ then } \zeta \text{ else } 0) \ x' + d\text{-OUT } j \ x')$
using $\zeta\text{-pos}$ **by**(intro d-OUT-add)
also have $d\text{-OUT } (\lambda e. (\text{if } e \in \text{set } E \text{ then } \zeta \text{ else } 0) \ x' = \int^+ y. \zeta * \text{indicator}$
 $(\text{set } E) \ (x', y) \ \partial \text{count-space UNIV}$

```

    unfolding d-OUT-def by(rule nn-integral-cong)(simp)
    also have ... = ( $\int^+ e. \zeta * \text{indicator (set } E) e \partial \text{embed-measure (count-space UNIV) (Pair } x')$ )
    by(simp add: measurable-embed-measure1 nn-integral-embed-measure)
    also have ... = ( $\int^+ e. \zeta * \text{indicator (set } [(x'', y) \leftarrow E. x'' = x'] e \partial \text{count-space UNIV)$ )
    by(auto simp add: embed-measure-count-space' nn-integral-count-space-indicator
    intro!: nn-integral-cong split: split-indicator)
    also have ... =  $\zeta * \text{card (set } [(x'', y) \leftarrow E. x'' = x'])$  using  $\zeta\text{-pos}$  by(simp
    add: nn-integral-cmult)
    finally show ?thesis .
  qed
  have  $IN\text{-}j'$ :  $d\text{-}IN \text{ ?}j' x' = \zeta * \text{card (set } [(y, x'') \leftarrow E. x'' = x']) + d\text{-}IN j x'$  for
   $x'$ 
  proof -
    have  $d\text{-}IN \text{ ?}j' x' = d\text{-}IN (\lambda e. \text{if } e \in \text{set } E \text{ then } \zeta \text{ else } 0) x' + d\text{-}IN j x'$ 
    using  $\zeta\text{-pos}$  by(intro d-IN-add)
    also have  $d\text{-}IN (\lambda e. \text{if } e \in \text{set } E \text{ then } \zeta \text{ else } 0) x' = \int^+ y. \zeta * \text{indicator (set } E) (y, x') \partial \text{count-space UNIV}$ 
    unfolding d-IN-def by(rule nn-integral-cong)(simp)
    also have ... = ( $\int^+ e. \zeta * \text{indicator (set } E) e \partial \text{embed-measure (count-space UNIV) } (\lambda y. (y, x'))$ )
    by(simp add: measurable-embed-measure1 nn-integral-embed-measure)
    also have ... = ( $\int^+ e. \zeta * \text{indicator (set } [(y, x'') \leftarrow E. x'' = x']) e \partial \text{count-space UNIV)$ 
    by(auto simp add: embed-measure-count-space' nn-integral-count-space-indicator
    intro!: nn-integral-cong split: split-indicator)
    also have ... =  $\zeta * \text{card (set } [(y, x'') \leftarrow E. x'' = x'])$ 
    using  $\zeta\text{-pos}$  by(auto simp add: nn-integral-cmult)
    finally show ?thesis .
  qed

  have  $j'$ :  $\text{flow } \Psi j'$ 
  proof
    fix  $e :: 'v \text{ option edge}$ 
    consider  $(None) e = (None, \text{Some } y)$ 
    |  $(\text{Some } x) y$  where  $e = (\text{Some } x, \text{Some } y) (x, y) \in \text{set } ?E$ 
    |  $(\text{old } x) y$  where  $e = (\text{Some } x, \text{Some } y) (x, y) \notin \text{set } ?E$ 
    |  $y'$  where  $e = (None, \text{Some } y') y \neq y'$ 
    |  $e = (None, None) | x$  where  $e = (\text{Some } x, None)$ 
    by(cases e; case-tac a; case-tac b)(auto)
    then show  $j' e \leq \text{capacity } \Psi e$  using  $uy \zeta\text{-pos flowD-capacity}[OF j, of e]$ 
    proof(cases)
      case None
      have  $\zeta \leq u y - j (None, \text{Some } y)$  by(simp add:  $\zeta\text{-def}$ )
      then have  $\zeta + j (None, \text{Some } y) \leq u y$ 
      using  $\zeta\text{-pos}$  by (auto simp add: ennreal-le-minus-iff)
      thus ?thesis using  $\text{reachable-}V[OF \text{ reach}] None \Psi.\text{flowD-outside}[OF j, of$ 
       $(\text{Some } y, None)] uy$ 

```

```

      by(auto simp add: j'-def E-def)
next
  case (Some x' y')
  have e:  $\zeta \leq \text{cap}'(x', y')$  using Some(2) by(rule  $\zeta\text{-le}$ )
  then consider (backward)  $\text{edge } \Gamma \ x' \ y' \ x' \neq a \mid$  (forward)  $\text{edge } \Gamma \ y' \ x' \neg$ 
 $\text{edge } \Gamma \ x' \ y'$ 
    | ( $a'$ )  $\text{edge } \Gamma \ x' \ y' \ x' = a$ 
    using Some  $\zeta\text{-pos}$  by(auto split: if-split-asm)
  then show ?thesis
  proof cases
    case backward
    have  $\zeta \leq f(x', y') + j(\text{Some } y', \text{Some } x') - j(\text{Some } x', \text{Some } y')$ 
      using e backward Some(1) by(simp add: g-simps)
    hence  $\zeta + j(\text{Some } x', \text{Some } y') - j(\text{Some } y', \text{Some } x') \leq (f(x', y') +$ 
 $j(\text{Some } y', \text{Some } x') - j(\text{Some } x', \text{Some } y')) + j(\text{Some } x', \text{Some } y') - j(\text{Some}$ 
 $y', \text{Some } x')$ 
      by(intro ennreal-minus-mono add-right-mono) simp-all
    also have  $\dots = f(x', y')$ 
      using j-le-f[OF  $\langle \text{edge } \Gamma \ x' \ y' \rangle$ ]
      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  $\zeta + j(\text{Some } x', \text{Some } y') - j(\text{Some } y', \text{Some } x') \leq \zeta + j(\text{Some } x',$ 
 $\text{Some } y')$ 
      by(rule diff-le-self-ennreal)
    also have  $j(\text{Some } x', \text{Some } y') \leq d\text{-IN } j(\text{Some } y')$ 
      by(rule d-IN-ge-point)
    also have  $\dots \leq \text{weight } \Gamma \ y'$  by(rule IN-j-le-weight)
    also have  $\zeta \leq 1$  using e forward by simp
    finally have  $\zeta + j(\text{Some } x', \text{Some } y') - j(\text{Some } y', \text{Some } x') \leq \max$ 
 $(\text{weight } \Gamma \ x') (\text{weight } \Gamma \ y') + 1$ 
      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  $a \in \text{set } (\text{map fst } (\text{zip } (y \# q) q))$  by(auto intro:
rev-image-eqI)
    also have  $\text{map fst } (\text{zip } (y \# q) q) = \text{butlast } (y \# q)$  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  $\zeta$ -pos
    by(auto simp add:  $j'$ -def  $E$ -def intro!: diff-le-self-ennreal)
    also have  $j e \leq \text{capacity } \Psi e$  using  $j$  by(rule flowD-capacity)
    finally show ?thesis .
qed(auto simp add:  $j'$ -def  $E$ -def  $\Psi$ .flowD-outside[OF  $j$ ] uy)
next
fix  $x'$ 
assume  $x'$ :  $x' \neq \text{source } \Psi$   $x' \neq \text{sink } \Psi$ 
then obtain  $x''$  where  $x''$ :  $x' = \text{Some } x''$  by auto
have  $d\text{-OUT } ?j' x' = \zeta * \text{card } (\text{set } [(x'', y) \leftarrow E. x'' = x']) + d\text{-OUT } j x'$ 
by(rule OUT- $j'$ )
also have  $\text{card } (\text{set } [(x'', y) \leftarrow E. x'' = x']) = \text{card } (\text{set } [(y, x'') \leftarrow E. x'' = x'])$  (is ?lhs = ?rhs)
proof -
  have ?lhs =  $\text{length } [(x'', y) \leftarrow E. x'' = x']$  using distinct
  by(subst distinct-card)(auto simp add:  $E$ -def filter-map distinct-map
inj-map-prod' distinct- $\text{zipI1}$ )
  also have  $\dots = \text{length } [x''' \leftarrow \text{map fst } ?E. x''' = x'']$ 
  by(simp add:  $E$ -def  $x''$  split-beta cong: filter-cong)
  also have  $\text{map fst } ?E = \text{butlast } (y \# q)$  by(induction  $q$  arbitrary:  $y$ ) simp-all
  also have  $[x''' \leftarrow \text{butlast } (y \# q). x''' = x''] = [x''' \leftarrow y \# q. x''' = x'']$ 
  using  $q$ -Nil rtranci-path-last[OF  $q$   $q$ -Nil]  $x' x''$ 
  by(cases  $q$  rule: rev-cases) simp-all
  also have  $q = \text{map snd } ?E$  by(induction  $q$  arbitrary:  $y$ ) auto
  also have  $\text{length } [x''' \leftarrow y \# \dots x''' = x''] = \text{length } [x'' \leftarrow \text{map snd } E. x'' = x']$  using  $x''$ 
  by(simp add:  $E$ -def cong: filter-cong)
  also have  $\dots = \text{length } [(y, x'') \leftarrow E. x'' = x']$  by(simp cong: filter-cong
add: split-beta)
  also have  $\dots = ?rhs$  using distinct
  by(subst distinct-card)(auto simp add:  $E$ -def filter-map distinct-map
inj-map-prod' distinct- $\text{zipI1}$ )
  finally show ?thesis .
qed
also have  $\zeta * \dots + d\text{-OUT } j x' = d\text{-IN } ?j' x'$ 
unfolding flowD-KIR[OF  $j$   $x'$ ] by(rule IN- $j'$ [symmetric])
also have  $d\text{-IN } ?j' x' \neq \top$ 
using  $\Psi$ .flowD-finite-IN[OF  $j$   $x'(2)$ ] finite- $\zeta$  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  $\text{value-flow } \Psi j' \leq \alpha$  unfolding  $\alpha$ -def by(auto intro: SUP-upper)
moreover have  $\text{value-flow } \Psi j' > \text{value-flow } \Psi j$ 
proof -
  have  $\text{value-flow } \Psi j + 0 < \text{value-flow } \Psi j + \zeta * 1$ 
  using  $\zeta$ -pos value- $j$  finite-flow by simp
  also have  $[(x', y') \leftarrow E. x' = \text{None}] = [(\text{None}, \text{Some } y)]$ 
  using  $q$ -Nil by(cases  $q$ )(auto simp add:  $E$ -def filter-map cong: filter-cong
split-beta)

```

hence $\zeta * 1 \leq \zeta * \text{card } (\text{set } [(x', y') \leftarrow E. x' = \text{None}])$ **using** $\zeta\text{-pos}$
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* $\Psi j + \dots = \text{value-flow } \Psi ?j'$
 using *OUT-j'* **by**(*simp add: add.commute*)
 also have $\dots = \text{value-flow } \Psi j'$ **unfolding** *j'-def*
by(*subst value-flow-cleanup*)(*auto simp add: E-def $\Psi.\text{flowD-outside}[OF j]$*)
 finally show *?thesis* **by**(*simp add: add-left-mono*)
qed
 ultimately show *False* **using** *finite-flow $\zeta\text{-pos}$ value-j*
by(*cases value-flow Ψj ζ rule: ennreal2-cases*) *simp-all*
qed

have *sep-h*: $y \in \text{TER } h$ **if** *reach*: *reachable* y **and** $y \in B \Gamma$ **and** *TER*: $y \in$
?TER f **for** y
proof(*rule ccontr*)
 assume $y': y \notin \text{TER } h$
 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*: $y \in \text{SINK } h$ **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*: $d\text{-IN } h y = d\text{-IN } g y$ **using** *reach* **by**(*simp add: IN-h*)
 also have $\dots = d\text{-IN } f y + j$ (*None, Some y*) **by**(*simp add: IN-g*)
 also have $d\text{-IN } f y = \text{weight } \Gamma y - u y$ **using** *currentD-weight-IN[OF f, of y]*
y disjoint TER
by(*auto elim!: SAT.cases*)
 also have $d\text{-IN } h y < \text{weight } \Gamma y$ **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 $\Gamma y u y j$ (*None, Some y*) rule: ennreal3-cases; cases $u y \leq$*
weight Γy)
 (*auto simp: ennreal-minus ennreal-plus[symmetric] add-top ennreal-less-iff*
ennreal-neg simp del: ennreal-plus)
 moreover from *reach* have j (*None, $\langle y \rangle$*) $= u y$ **by**(*rule reachable-full*)
 ultimately show *False* **by** *simp*
qed

have w' : *wave* Γh
proof
 show *sep*: *separating* Γ (*TER* h)
proof(*rule ccontr*)
 assume $\neg ?thesis$
 then obtain $x p y$ **where** x : $x \in A \Gamma$ **and** y : $y \in B \Gamma$ **and** p : *path* $\Gamma x p y$

and x' : $x \notin \text{TER } h$ **and** *bypass*: $\bigwedge z. z \in \text{set } p \implies z \notin \text{TER } h$
by(*auto simp add: separating-gen.simps*)
from p *disjoint* x y **have** $p\text{-eq}$: $p = [y]$ **and** *edge*: $(x, y) \in \mathbf{E}$
by $-(\text{erule } r\text{trancl-path.cases}, \text{auto dest: bipartite-E})+$
from $p\text{-eq}$ *bypass* **have** y' : $y \notin \text{TER } h$ **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 $*$: $x \notin \mathcal{E}_{\text{?}\Gamma} (\text{?TER } f)$ **using** x'
proof(*rule contrapos-nn*)
assume $*$: $x \in \mathcal{E}_{\text{?}\Gamma} (\text{?TER } f)$
have $d\text{-OUT } h \ x \leq d\text{-OUT } g \ x$ **using** $h\text{-le-}g$ **by**(*rule d-OUT-mono*)
also from $*$ **have** $x \neq a$ **using** a **by** *auto*
then have $d\text{-OUT } j \ (\text{Some } x) = d\text{-IN } j \ (\text{Some } x)$ **by**(*auto intro: flowD-KIR[OF*
j])
hence $d\text{-OUT } g \ x \leq d\text{-OUT } f \ x$ **using** $\text{OUT-}g\text{-}A[\text{OF } x]$ $\text{IN-}j[\text{of } \text{Some } x]$
finite-flow
by(*auto split: if-split-asm*)
also have $\dots = 0$ **using** $*$ **by**(*auto elim: SINK.cases*)
finally have $x \in \text{SINK } h$ **by**(*simp add: SINK.simps*)
with x **show** $x \in \text{TER } h$ **by**(*simp add: SAT.A*)
qed
from p $p\text{-eq}$ x y **have** *path* $\text{?}\Gamma \ x \ [y] \ y \ x \in A \ \text{?}\Gamma \ y \in B \ \text{?}\Gamma$ **by** *simp-all*
from $*$ *separatingD*[*OF separating-essential, OF waveD-separating, OF w this*]
have $y \in \text{?TER } f$ **by** *auto*
with *reach* y **have** $y \in \text{TER } h$ **by**(*rule sep-h*)
with y' **show** *False* **by** *contradiction*
qed
qed(*rule h*)

have $\text{OUT-}g\text{-}a$: $d\text{-OUT } g \ a = d\text{-OUT } h \ a$ **by**(*simp add: OUT-h*)
have $a \notin \mathcal{E} (\text{TER } h)$
proof
assume $*$: $a \in \mathcal{E} (\text{TER } h)$

have $j \ (\text{Some } a, \text{Some } y) = 0$ **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 $d\text{-OUT } f \ a \leq d\text{-OUT } g \ a$ **unfolding** $d\text{-OUT-def}$
 — 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 $a \in \text{SINK } f$ **using** $\text{OUT-}g\text{-}a$ $*$ **by**(*simp add: SINK.simps*)
with $a(1)$ **have** $a \in \text{?TER } f$ **by**(*auto intro: SAT.A*)
with $a(2)$ **have** a' : $\neg \text{essential } \Gamma \ (B \ \Gamma) \ (\text{?TER } f) \ a$ **by** *simp*

from $*$ **obtain** y **where** ay : *edge* $\Gamma \ a \ y$ **and** y : $y \in B \ \Gamma$ **and** y' : $y \notin \text{TER } h$


```

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:  $y \in ?TER$  by simp

  have reachable y using ‹reachable a› by(rule reachable-AB)(simp add: ay)
  hence  $y \in TER$  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 +  $\int^+ y. j$  (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 ( $\int^+ y. j$  (Some y, Some a) ∂count-space UNIV) = ( $\int^+ y. j$  (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 ≡  $\sum^+_{x \in B \Gamma} d-IN g x - d-OUT g x$ 
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( $\text{weight} := \lambda x. \text{if } x \in A \Gamma \text{ then } \text{weight } \Gamma x - d-OUT g x \text{ else } \text{weight } \Gamma x$ )
  have Γ'-sel [simp]:
    edge Γ' = edge Γ
    A Γ' = A Γ

```

```

    B  $\Gamma'$  = B  $\Gamma$ 
    weight  $\Gamma' x$  = (if  $x \in A \Gamma$  then weight  $\Gamma x - d\text{-OUT } g x$  else weight  $\Gamma x$ )
    vertex  $\Gamma' = \text{vertex } \Gamma$ 
    web.more  $\Gamma' = \text{web.more } \Gamma$ 
    for  $x$  by(simp-all add:  $\Gamma'$ -def)
    have countable-bipartite-web  $\Gamma'$ 
      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  $\Gamma'$ : countable-bipartite-web  $\Gamma'$  .
    let  $?u = \lambda x. (d\text{-IN } g x - d\text{-OUT } g x) * \text{indicator } (- A \Gamma) x$ 

    have hindered  $\Gamma'$ 
    proof(rule  $\Gamma'$ .hindered-reduce)
      show  $?u x = 0$  if  $x \notin B \Gamma'$  for  $x$  using that bipartite-V
      by(cases vertex  $\Gamma' x$ )(auto simp add: currentD-outside-OUT[OF g] currentD-outside-IN[OF g])

      have *:  $(\sum^+ x \in B \Gamma'. ?u x) = \varepsilon$  using disjoint
      by(auto intro!: nn-integral-cong simp add:  $\varepsilon$ -def nn-integral-count-space-indicator
currentD-outside-OUT[OF g] currentD-outside-IN[OF g] not-vertex split: split-indicator)
      thus  $(\sum^+ x \in B \Gamma'. ?u x) \neq \top$  using  $\varepsilon$ -finite by simp

      have **:  $\Gamma'(\text{weight} := \text{weight } \Gamma' - ?u) = \Gamma \ominus g$ 
      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  $(\Gamma'(\text{weight} := \text{weight } \Gamma' - ?u)) (\sum^+ x \in B \Gamma'. ?u x)$ 
      unfolding * ** by(fact hindered)
      show  $(\lambda x. (d\text{-IN } g x - d\text{-OUT } g x) * \text{indicator } (- A \Gamma) x) \leq \text{weight } \Gamma'$ 
      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
end

```

10.3 Reduced weight in a loose web

definition *reduce-weight* :: $('v, 'more) \text{ web-scheme} \Rightarrow 'v \Rightarrow \text{real} \Rightarrow ('v, 'more) \text{ web-scheme}$

where *reduce-weight* $\Gamma x r = \Gamma(\text{weight} := \lambda y. \text{weight } \Gamma y - (\text{if } x = y \text{ then } r \text{ else } 0))$

lemma *reduce-weight-sel* [simp]:

edge (*reduce-weight* $\Gamma x r$) = *edge* Γ

A (*reduce-weight* $\Gamma x r$) = *A* Γ

$B \text{ (reduce-weight } \Gamma \ x \ r) = B \ \Gamma$
 $\text{vertex (reduce-weight } \Gamma \ x \ r) = \text{vertex } \Gamma$
 $\text{weight (reduce-weight } \Gamma \ x \ r) \ y = (\text{if } x = y \text{ then weight } \Gamma \ x - r \text{ else weight } \Gamma \ y)$
 $\text{web.more (reduce-weight } \Gamma \ x \ r) = \text{web.more } \Gamma$
by(simp-all add: reduce-weight-def zero-ennreal-def[symmetric] vertex-def fun-eq-iff)

lemma essential-reduce-weight [simp]: essential (reduce-weight $\Gamma \ x \ r$) = essential Γ
by(simp add: fun-eq-iff essential-def)

lemma roofed-reduce-weight [simp]: roofed-gen (reduce-weight $\Gamma \ x \ r$) = 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: $x \notin \text{range Inner} \longleftrightarrow x = \text{SOURCE} \vee x = \text{SINK}$
by(cases x) auto

private lemma inj-Inner [simp]: $\bigwedge A. \text{inj-on Inner } A$
by(simp add: inj-on-def)

lemma unhinder-bipartite:

assumes $h: \bigwedge n :: \text{nat. current } \Gamma \ (h \ n)$
and $\text{SAT}: \bigwedge n. (B \ \Gamma \cap \mathbf{V}) - \{b\} \subseteq \text{SAT } \Gamma \ (h \ n)$
and $b: b \in B \ \Gamma$
and $\text{IN}: (\text{SUP } n. d\text{-IN } (h \ n) \ b) = \text{weight } \Gamma \ b$
and $h0\text{-}b: \bigwedge n. d\text{-IN } (h \ 0) \ b \leq d\text{-IN } (h \ n) \ b$
and $b\text{-}V: b \in \mathbf{V}$
shows $\exists h'. \text{current } \Gamma \ h' \wedge \text{wave } \Gamma \ h' \wedge B \ \Gamma \cap \mathbf{V} \subseteq \text{SAT } \Gamma \ h'$

proof –

write Inner ($\langle \langle - \rangle \rangle$)

define edge'

where edge' $xo \ yo =$
 (case (xo, yo) of
 $(\langle x \rangle, \langle y \rangle) \Rightarrow \text{edge } \Gamma \ x \ y \vee \text{edge } \Gamma \ y \ x$
 $| (\langle x \rangle, \text{SINK}) \Rightarrow x \in A \ \Gamma$
 $| (\text{SOURCE}, \langle y \rangle) \Rightarrow y = b$
 $| (\text{SINK}, \langle x \rangle) \Rightarrow x \in A \ \Gamma$
 $| - \Rightarrow \text{False}$) **for** xo yo

have edge'-simps [simp]:

$\text{edge}' \ \langle x \rangle \ \langle y \rangle \longleftrightarrow \text{edge } \Gamma \ x \ y \vee \text{edge } \Gamma \ y \ x$
 $\text{edge}' \ \langle x \rangle \ \text{SINK} \longleftrightarrow x \in A \ \Gamma$
 $\text{edge}' \ \text{SOURCE} \ y \longleftrightarrow y = \langle b \rangle$

```

    edge' SINK  $\langle x \rangle \longleftrightarrow x \in A \Gamma$ 
    edge' SINK SINK  $\longleftrightarrow \text{False}$ 
    edge' xo SOURCE  $\longleftrightarrow \text{False}$ 
    for x y yo xo by(simp-all add: edge'-def split: vertex.split)
  have edge'E: thesis if edge' xo yo
   $\wedge x y. \llbracket xo = \langle x \rangle; yo = \langle y \rangle; \text{edge } \Gamma x y \vee \text{edge } \Gamma y x \rrbracket \implies \text{thesis}$ 
   $\wedge x. \llbracket xo = \langle x \rangle; yo = \text{SINK}; x \in A \Gamma \rrbracket \implies \text{thesis}$ 
   $\wedge x. \llbracket xo = \text{SOURCE}; yo = \langle b \rangle \rrbracket \implies \text{thesis}$ 
   $\wedge y. \llbracket xo = \text{SINK}; yo = \langle y \rangle; y \in A \Gamma \rrbracket \implies \text{thesis}$ 
  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
   $\wedge y. \llbracket yo = \langle y \rangle; \text{edge } \Gamma x y \vee \text{edge } \Gamma y x \rrbracket \implies \text{thesis}$ 
   $\llbracket yo = \text{SINK}; x \in A \Gamma \rrbracket \implies \text{thesis}$ 
  for x yo thesis using that by(auto elim: edge'E)
  have edge'-Inner2 [elim!]: thesis if edge' xo  $\langle y \rangle$ 
   $\wedge x. \llbracket xo = \langle x \rangle; \text{edge } \Gamma x y \vee \text{edge } \Gamma y x \rrbracket \implies \text{thesis}$ 
   $\llbracket xo = \text{SOURCE}; y = b \rrbracket \implies \text{thesis}$ 
   $\llbracket xo = \text{SINK}; y \in A \Gamma \rrbracket \implies \text{thesis}$ 
  for xo y thesis using that by(auto elim: edge'E)
  have edge'-SINK1 [elim!]: thesis if edge' SINK yo
   $\wedge y. \llbracket yo = \langle y \rangle; y \in A \Gamma \rrbracket \implies \text{thesis}$ 
  for yo thesis using that by(auto elim: edge'E)
  have edge'-SINK2 [elim!]: thesis if edge' xo SINK
   $\wedge x. \llbracket xo = \langle x \rangle; x \in A \Gamma \rrbracket \implies \text{thesis}$ 
  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  $\Gamma x y$  then  $h \ 0 \ (x, y)$  else if edge  $\Gamma y x$  then  $\max \ (\text{weight } \Gamma x) \ (\text{weight } \Gamma y)$  else 0
    | ( $\langle x \rangle, \text{SINK}$ )  $\Rightarrow$  if  $x \in A \Gamma$  then  $\text{weight } \Gamma x - d\text{-OUT } (h \ 0) \ x$  else 0
    | ( $\text{SOURCE}, yo$ )  $\Rightarrow$  if  $yo = \langle b \rangle$  then  $\text{weight } \Gamma b - d\text{-IN } (h \ 0) \ b$  else 0
    | ( $\text{SINK}, \langle y \rangle$ )  $\Rightarrow$  if  $y \in A \Gamma$  then  $\text{weight } \Gamma y$  else 0
    | -  $\Rightarrow 0$ ) for xoyo
  have cap-simps [simp]:
    cap ( $\langle x \rangle, \langle y \rangle$ ) = (if edge  $\Gamma x y$  then  $h \ 0 \ (x, y)$  else if edge  $\Gamma y x$  then  $\max \ (\text{weight } \Gamma x) \ (\text{weight } \Gamma y)$  else 0)
    cap ( $\langle x \rangle, \text{SINK}$ ) = (if  $x \in A \Gamma$  then  $\text{weight } \Gamma x - d\text{-OUT } (h \ 0) \ x$  else 0)
    cap ( $\text{SOURCE}, yo$ ) = (if  $yo = \langle b \rangle$  then  $\text{weight } \Gamma b - d\text{-IN } (h \ 0) \ b$  else 0)
    cap ( $\text{SINK}, \langle y \rangle$ ) = (if  $y \in A \Gamma$  then  $\text{weight } \Gamma y$  else 0)
    cap ( $\text{SINK}, \text{SINK}$ ) = 0
    cap (xo, SOURCE) = 0
  for x y yo xo by(simp-all add: cap-def split: vertex.split)
define  $\Psi$  where  $\Psi = (\text{edge} = \text{edge}', \text{capacity} = \text{cap}, \text{source} = \text{SOURCE}, \text{sink} = \text{SINK})$ 
  have  $\Psi\text{-sel}$  [simp]:
    edge  $\Psi = \text{edge}'$ 

```

```

capacity  $\Psi = \text{cap}$ 
source  $\Psi = \text{SOURCE}$ 
sink  $\Psi = \text{SINK}$ 
by(simp-all add:  $\Psi$ -def)

have cap-outside1:  $\neg \text{vertex } \Gamma x \implies \text{cap } (\langle x \rangle, y) = 0$  for  $x y$  using A-vertex
by(cases y)(auto simp add: vertex-def)
have capacity-A-weight:  $d\text{-OUT } \text{cap } \langle x \rangle \leq 2 * \text{weight } \Gamma x$  if  $x \in A \cap \Gamma$  for  $x$ 
proof -
  have  $d\text{-OUT } \text{cap } \langle x \rangle \leq (\sum^+ y. h\ 0\ (x, \text{inner } y) * \text{indicator } (\text{range } \text{Inner})\ y$ 
+  $\text{weight } \Gamma x * \text{indicator } \{\text{SINK}\}\ y)$ 
  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  $\dots = (\sum^+ y \in \text{range } \text{Inner}. h\ 0\ (x, \text{inner } y)) + \text{weight } \Gamma x$ 
  by(auto simp add: nn-integral-count-space-indicator nn-integral-add)
  also have  $(\sum^+ y \in \text{range } \text{Inner}. h\ 0\ (x, \text{inner } y)) = d\text{-OUT } (h\ 0)\ x$ 
  by(simp add: d-OUT-def nn-integral-count-space-reindex)
  also have  $\dots \leq \text{weight } \Gamma x$  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  $\Psi$ 
proof
  have  $\mathbf{E}_\Psi \subseteq (\lambda(x, y). (\langle x \rangle, \langle y \rangle)) \text{ ' } \mathbf{E} \cup (\lambda(x, y). (\langle y \rangle, \langle x \rangle)) \text{ ' } \mathbf{E} \cup (\lambda x. (\langle x \rangle,$ 
SINK)  $\text{ ' } A \cap \Gamma \cup (\lambda x. (\text{SINK}, \langle x \rangle)) \text{ ' } A \cap \Gamma \cup \{(\text{SOURCE}, \langle b \rangle)\}$ 
  by(auto simp add: edge'-def split: vertex.split-asm)
  moreover have countable  $(A \cap \Gamma)$  using A-vertex by(rule countable-subset) simp
  ultimately show countable  $\mathbf{E}_\Psi$  by(auto elim: countable-subset)
next
  fix  $v$ 
  assume  $v \neq \text{sink } \Psi$ 
  then consider (source)  $v = \text{SOURCE} \mid (A)\ x$  where  $v = \langle x \rangle\ x \in A \cap \Gamma$ 
 $\mid (B)\ y$  where  $v = \langle y \rangle\ y \notin A \cap \Gamma\ y \in \mathbf{V} \mid (\text{outside})\ x$  where  $v = \langle x \rangle\ x \notin \mathbf{V}$ 
  by(cases v) auto
  then show  $d\text{-IN } (\text{capacity } \Psi)\ v \neq \top \vee d\text{-OUT } (\text{capacity } \Psi)\ v \neq \top$ 
  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 en-
nreal-mult-eq-top-iff)
  next
    case  $(B\ y)$ 
    have  $d\text{-IN } (\text{capacity } \Psi)\ v \leq (\sum^+ x. h\ 0\ (\text{inner } x, y) * \text{indicator } (\text{range } \text{Inner})\ x$ 
+  $\text{weight } \Gamma b * \text{indicator } \{\text{SOURCE}\}\ x)$ 
    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 ... = ( $\sum^+ x \in \text{range } \text{Inner}. h\ 0\ (\text{inner } x, y) + \text{weight } \Gamma\ b$ 
    by(simp add: nn-integral-add nn-integral-count-space-indicator)
  also have ( $\sum^+ x \in \text{range } \text{Inner}. h\ 0\ (\text{inner } x, y) = d\text{-IN } (h\ 0)\ y$ 
    by(simp add: d-IN-def nn-integral-count-space-reindex)
  also have  $d\text{-IN } (h\ 0)\ y \leq \text{weight } \Gamma\ y$  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  $d\text{-OUT } (\text{capacity } \Psi)\ v = 0$  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  $\Psi\ e \neq \top$  for e
    by(auto simp add: cap-def max-def vertex-def currentD-finite[OF h] split:
vertex.split prod.split)
  show capacity  $\Psi\ e = 0$  if  $e \notin E_\Psi$  for e using that
    by(auto simp add: cap-def max-def split: prod.split; split vertex.split)+
  show  $\neg \text{edge } \Psi\ x\ (\text{source } \Psi)$  for x using b by(auto simp add: B-out)
  show  $\neg \text{edge } \Psi\ x\ x$  for x by(cases x)(simp-all add: no-loop)
  show  $\text{source } \Psi \neq \text{sink } \Psi$  by simp
qed
then interpret  $\Psi$ : flow-attainability  $\Psi$  .
define  $\alpha$  where  $\alpha = (\text{SUP } f \in \{f. \text{flow } \Psi\ f\}. \text{value-flow } \Psi\ f)$ 

define f
  where  $f\ n\ \text{xoyo} =$ 
    (case xoyo of
      | ( $\langle x \rangle, \langle y \rangle$ )  $\Rightarrow$  if edge  $\Gamma\ x\ y$  then  $h\ 0\ (x, y) - h\ n\ (x, y)$  else if edge  $\Gamma\ y\ x$  then
 $h\ n\ (y, x) - h\ 0\ (y, x)$  else 0
      | ( $\text{SOURCE}, \langle y \rangle$ )  $\Rightarrow$  if  $y = b$  then  $d\text{-IN } (h\ n)\ b - d\text{-IN } (h\ 0)\ b$  else 0
      | ( $\langle x \rangle, \text{SINK}$ )  $\Rightarrow$  if  $x \in A\ \Gamma$  then  $d\text{-OUT } (h\ n)\ x - d\text{-OUT } (h\ 0)\ x$  else 0
      | ( $\text{SINK}, \langle y \rangle$ )  $\Rightarrow$  if  $y \in A\ \Gamma$  then  $d\text{-OUT } (h\ 0)\ y - d\text{-OUT } (h\ n)\ y$  else 0
      | -  $\Rightarrow 0$ ) for n xoyo
  have f-cases: thesis if  $\bigwedge x\ y. e = (\langle x \rangle, \langle y \rangle) \Longrightarrow \text{thesis } \bigwedge y. e = (\text{SOURCE}, \langle y \rangle)$ 
 $\Longrightarrow \text{thesis}$ 
     $\bigwedge x. e = (\langle x \rangle, \text{SINK}) \Longrightarrow \text{thesis } \bigwedge y. e = (\text{SINK}, \langle y \rangle) \Longrightarrow \text{thesis } e = (\text{SINK},$ 
 $\text{SINK}) \Longrightarrow \text{thesis}$ 
     $\bigwedge xo. e = (xo, \text{SOURCE}) \Longrightarrow \text{thesis } e = (\text{SOURCE}, \text{SINK}) \Longrightarrow \text{thesis}$ 
    for  $e :: 'v\ \text{vertex edge}$  and thesis
    using that by(cases e; cases fst e snd e rule: vertex.exhaust[case-product ver-
tex.exhaust]) simp-all
  have f-simps [simp]:
     $f\ n\ (\langle x \rangle, \langle y \rangle) = (\text{if edge } \Gamma\ x\ y \text{ then } h\ 0\ (x, y) - h\ n\ (x, y) \text{ else if edge } \Gamma\ y\ x$ 
then  $h\ n\ (y, x) - h\ 0\ (y, x)$  else 0)
     $f\ n\ (\text{SOURCE}, \langle y \rangle) = (\text{if } y = b \text{ then } d\text{-IN } (h\ n)\ b - d\text{-IN } (h\ 0)\ b \text{ else } 0)$ 

```

```

  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 = (∑+ 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] ver-
tex-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: en-
nreal-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: (∑+ 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)

```



```

 $x - (\sum^+ y. h\ 0\ (x, y) * indicator\ ?h0-ge-hn\ y)) + f\ n\ (SINK, \langle x \rangle)$ 
  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  $d-OUT\ (h\ 0)\ x - (\sum^+ y. h\ 0\ (x, y) * indicator\ ?h0-ge-hn\ y) =$ 
 $(\sum^+ y. h\ 0\ (x, y) * indicator\ ?h0-lt-hn\ y)$ 
    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  $(\sum^+ y. h\ n\ (x, y) * indicator\ ?h0-lt-hn\ y) - \dots = (\sum^+ y. h\ n$ 
 $(x, y) - h\ 0\ (x, y))$ 
      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  $\dots = (\sum^+ y \in range\ Inner. f\ n\ (y, \langle x \rangle))$  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  $\dots + f\ n\ (SINK, \langle x \rangle) = (\sum^+ y. f\ n\ (y, \langle x \rangle) * indicator\ (range$ 
 $Inner)\ y + f\ n\ (y, \langle x \rangle) * indicator\ \{SINK\}\ y)$ 
        by (simp add: nn-integral-add nn-integral-count-space-indicator)
      also have  $\dots = d-IN\ (f\ n)\ \langle x \rangle$ 
        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:  $d\text{-}IN\ (h\ 0)\ x + f\ n\ (SOURCE, \langle x \rangle) = d\text{-}IN\ (h\ n)\ x$ 
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  $d\text{-}IN\ (h\ n)\ x = d\text{-}IN\ (h\ 0)\ x$  by(auto simp add:
currentD-SAT[OF  $h$ ])
  thus ?thesis using False by(simp add: currentD-finite-IN[OF  $h$ ])
qed

have  $d\text{-}IN\ (f\ n)\ \langle x \rangle = (\sum^+ y. f\ n\ (y, \langle x \rangle) * indicator\ (range\ Inner)\ y + f\ n\ (y, \langle x \rangle) * indicator\ \{SOURCE\}\ y)$ 
  using B disjoint unfolding d-IN-def
  by(intro nn-integral-cong)(auto split: split-indicator simp add: notin-range-Inner)
  also have  $\dots = (\sum^+ y \in range\ Inner. f\ n\ (y, \langle x \rangle)) + f\ n\ (SOURCE, \langle x \rangle)$ 
using h0-b[of  $n$ ]
  by(simp add: nn-integral-add nn-integral-count-space-indicator max-def)
  also have  $(\sum^+ y \in range\ Inner. f\ n\ (y, \langle x \rangle)) = (\sum^+ y. h\ 0\ (y, x) - h\ n\ (y, x))$ 
  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  $\dots = (\sum^+ y. h\ 0\ (y, x) * indicator\ ?h0\text{-}gt\text{-}hn\ y) - (\sum^+ y. h\ n\ (y, x) * indicator\ ?h0\text{-}gt\text{-}hn\ y)$ 
  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:  $(\sum^+ y. h\ 0\ (y, x) * indicator\ ?h0\text{-}gt\text{-}hn\ y) = d\text{-}IN\ (h\ 0)\ x - (\sum^+ y. h\ 0\ (y, x) * indicator\ ?h0\text{-}le\text{-}hn\ y)$ 
  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 eq-h-n:  $(\sum^+ y. h\ n\ (y, x) * indicator\ ?h0-gt-hn\ y) = d-IN\ (h\ n)$ 
 $x - (\sum^+ y. h\ n\ (y, x) * indicator\ ?h0-le-hn\ y)$ 
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 d-IN (h 0)  $x - (\sum^+ y. h\ 0\ (y, x) * indicator\ ?h0-le-hn\ y) - (d-IN$ 
 $(h\ n)\ x - (\sum^+ y. h\ n\ (y, x) * indicator\ ?h0-le-hn\ y)) + f\ n\ (SOURCE, \langle x \rangle) =$ 
 $(\sum^+ y. h\ n\ (y, x) * indicator\ ?h0-le-hn\ y) - (\sum^+ y. h\ 0\ (y, x) *$ 
 $indicator\ ?h0-le-hn\ y)$ 
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  $(\sum^+ y. h\ n\ (y, x) * indicator\ ?h0-le-hn\ y) - (\sum^+ y. h\ 0\ (y, x)$ 
 $* indicator\ ?h0-le-hn\ y) = (\sum^+ y. h\ n\ (y, x) - h\ 0\ (y, x))$ 
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  $\dots = (\sum^+ y \in range\ Inner. f\ n\ (\langle x \rangle, y))$  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  $\dots = (\sum^+ y. f\ n\ (\langle x \rangle, y) * indicator\ (range\ Inner)\ y)$ 
by(simp add: nn-integral-add nn-integral-count-space-indicator max.left-commute
max.commute)
also have  $\dots = d-OUT\ (f\ n)\ \langle x \rangle$  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 $\text{weight } \Gamma \ b - d\text{-IN } (h \ 0) \ b = (\text{SUP } n. \text{value-flow } \Psi \ (f \ n))$
 using $\text{OUT-f-SOURCE currentD-finite-IN}[OF \ h, \text{ of } 0 \ b] \text{ IN}$
 by $(\text{simp add: SUP-diff-ennreal less-top})$
 also have $(\text{SUP } n. \text{value-flow } \Psi \ (f \ n)) \leq \alpha$ **unfolding** $\alpha\text{-def}$
 apply (rule SUP-least)
 apply (rule SUP-upper)
 apply $(\text{simp add: } f)$
 done
 also have $\alpha \leq \text{weight } \Gamma \ b - d\text{-IN } (h \ 0) \ b$ **unfolding** $\alpha\text{-def}$
 proof $(\text{rule SUP-least; clarsimp})$
 fix f
 assume $f: \text{flow } \Psi \ f$
 have $d\text{-OUT } f \text{ SOURCE} = (\sum^+ y. f \ (\text{SOURCE}, y) * \text{indicator } \{\langle b \rangle\} \ y)$
unfolding $d\text{-OUT-def}$
 apply $(\text{rule nn-integral-cong})$
 subgoal for x using $\text{flowD-capacity}[OF \ f, \text{ of } (\text{SOURCE}, x)]$
 by $(\text{auto split: split-indicator})$
 done
 also have $\dots = f \ (\text{SOURCE}, \langle b \rangle)$ **by** $(\text{simp add: max-def})$
 also have $\dots \leq \text{weight } \Gamma \ b - d\text{-IN } (h \ 0) \ b$ **using** $\text{flowD-capacity}[OF \ f, \text{ of } (\text{SOURCE}, \langle b \rangle)]$ **by** simp
 finally show $d\text{-OUT } f \text{ SOURCE} \leq \dots$.
 qed
 ultimately have $\alpha: \alpha = \text{weight } \Gamma \ b - d\text{-IN } (h \ 0) \ b$ **by** $(\text{rule antisym}[rotated])$
 hence $\alpha\text{-finite: } \alpha \neq \top$ **by** simp

from $\Psi.\text{ex-max-flow}$
 obtain g where $g: \text{flow } \Psi \ g$
 and $\text{value-g: value-flow } \Psi \ g = \alpha$
 and $\text{IN-g: } \bigwedge x. d\text{-IN } g \ x \leq \text{value-flow } \Psi \ g$ **unfolding** $\alpha\text{-def}$ **by** blast

have $g\text{-le-h0: } g \ (\langle x \rangle, \langle y \rangle) \leq h \ 0 \ (x, y)$ **if** $\text{edge } \Gamma \ x \ y$ **for** $x \ y$
 using $\text{flowD-capacity}[OF \ g, \text{ of } (\langle x \rangle, \langle y \rangle)]$ **that** **by** simp
 note $[\text{simp}] = \Psi.\text{flowD-finite}[OF \ g]$

have $g\text{-SOURCE: } g \ (\text{SOURCE}, \langle x \rangle) = (\text{if } x = b \text{ then } \alpha \text{ else } 0)$ **for** x
 proof $(\text{cases } x = b)$
 case True
 have $g \ (\text{SOURCE}, \langle x \rangle) = (\sum^+ y. g \ (\text{SOURCE}, y) * \text{indicator } \{\langle x \rangle\} \ y)$ **by** $(\text{simp add: max-def})$
 also have $\dots = \text{value-flow } \Psi \ g$ **unfolding** $d\text{-OUT-def}$ **using** True
 by $(\text{intro nn-integral-cong})(\text{auto split: split-indicator intro: } \Psi.\text{flowD-outside}[OF \ g])$
 finally show $?thesis$ **using** value-g **by** (simp add: True)
 qed $(\text{simp add: } \Psi.\text{flowD-outside}[OF \ g])$

let $?g = \lambda(x, y). g \ (\langle y \rangle, \langle x \rangle)$

define h' **where** $h' = h \ 0 \oplus ?g$
have h' -simps: $h' \ (x, y) = (\text{if edge } \Gamma \ x \ y \text{ then } h \ 0 \ (x, y) + g \ (\langle y \rangle, \langle x \rangle) - g \ (\langle x \rangle, \langle y \rangle) \text{ else } 0)$ **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** $e \notin \mathbf{E}$ **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** $x \notin \mathbf{V}$ **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 : $g \ (SINK, \langle x \rangle) \leq d$ - $OUT \ g \ \langle x \rangle$ **for** x
by (subst flowD-KIR[OF g]) (simp-all add: d - IN -ge-point)

have OUT - g - A : d - $OUT \ ?g \ x = d$ - $OUT \ g \ \langle x \rangle - g \ (SINK, \langle x \rangle)$ **if** $x \in A \ \Gamma$ **for** x
proof –
have d - $OUT \ ?g \ x = (\sum^+ y \in \text{range } Inner. g \ (y, \langle x \rangle))$
by(simp add: nn-integral-count-space-reindex d - OUT -def)
also have $\dots = d$ - $IN \ g \ \langle x \rangle - (\sum^+ y. g \ (y, \langle x \rangle) * \text{indicator } \{SINK\} \ y)$
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 $\dots = d$ - $OUT \ g \ \langle x \rangle - g \ (SINK, \langle x \rangle)$ **by**(simp add: flowD-KIR[OF g] max-def)
finally show ?thesis .
qed

have IN - g - A : d - $IN \ ?g \ x = d$ - $OUT \ g \ \langle x \rangle - g \ (\langle x \rangle, SINK)$ **if** $x \in A \ \Gamma$ **for** x
proof –
have d - $IN \ ?g \ x = (\sum^+ y \in \text{range } Inner. g \ (\langle x \rangle, y))$
by(simp add: nn-integral-count-space-reindex d - IN -def)
also have $\dots = d$ - $OUT \ g \ \langle x \rangle - (\sum^+ y. g \ (\langle x \rangle, y) * \text{indicator } \{SINK\} \ y)$
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 $\dots = d$ - $OUT \ g \ \langle x \rangle - g \ (\langle x \rangle, SINK)$ **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 = (∑+ y∈range Inner. g (y, ⟨x⟩))
    by(simp add: nn-integral-count-space-reindex d-OUT-def)
    also have ... = d-IN g ⟨x⟩ - (∑+ 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 = (∑+ 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 = (∑+ 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 + (∑+ y. ?g (x, y) * indicator E (x, y))
    - (∑+ 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 (∑+ 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 $\dots = d\text{-OUT } g \langle x \rangle - g (SINK, \langle x \rangle)$ **using that** **by**(rule OUT-g-A)
also have $(\sum^+ y. ?g (y, x) * \text{indicator } \mathbf{E} (x, y)) = d\text{-IN } ?g x$ **using that**
unfolding d-IN-def
by(auto simp add: A-in split: split-indicator intro!: nn-integral-cong $\Psi.\text{flowD-outside}[OF$
 $g]$)
also have $\dots = d\text{-OUT } g \langle x \rangle - g (\langle x \rangle, SINK)$ **using that** **by**(rule IN-g-A)
also have $d\text{-OUT } (h \ 0) x + (d\text{-OUT } g \langle x \rangle - g (SINK, \langle x \rangle)) - \dots = d\text{-OUT}$
 $(h \ 0) x + g (\langle x \rangle, SINK) - g (SINK, \langle x \rangle)$
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: $\Psi.\text{flowD-finite-OUT}[OF g]$)
done
finally show ?thesis .
qed

have finite-g-OUT: $d\text{-OUT } ?g x \neq \top$ **for** x **using** $\alpha\text{-finite}$
proof(rule neq-top-trans)
have $d\text{-OUT } ?g x = (\sum^+ y \in \text{range } \text{Inner}. g (y, \langle x \rangle))$
by(auto simp add: d-OUT-def nn-integral-count-space-reindex)
also have $\dots \leq d\text{-IN } g \langle x \rangle$ **unfolding** d-IN-def
by(auto simp add: nn-integral-count-space-indicator intro!: nn-integral-mono
split: split-indicator)
also have $\dots \leq \alpha$ **using** IN-g value-g **by** simp
finally show $d\text{-OUT } ?g x \leq \alpha$.
qed

have IN-h'-B: $d\text{-IN } h' x = d\text{-IN } (h \ 0) x + g (SOURCE, \langle x \rangle)$ **if** $x \in B \ \Gamma$ **for** x
proof –
have g-le: $g (SOURCE, \langle x \rangle) \leq d\text{-IN } g \langle x \rangle$
by (rule d-IN-ge-point)

have $d\text{-IN } h' x = d\text{-IN } (h \ 0) x + (\sum^+ y. g (\langle x \rangle, \langle y \rangle) * \text{indicator } \mathbf{E} (y, x)) -$
 $(\sum^+ y. g (\langle y \rangle, \langle x \rangle) * \text{indicator } \mathbf{E} (y, x))$
unfolding h'-def
by(subst IN-plus-flow[of $\Gamma \ h \ 0 \ ?g$, OF currentD-outside'[OF h]])
(auto simp add: g-le-h0 finite-g-OUT)
also have $(\sum^+ y. g (\langle x \rangle, \langle y \rangle) * \text{indicator } \mathbf{E} (y, x)) = d\text{-IN } ?g x$ **unfolding**
d-IN-def **using that**
by(intro nn-integral-cong)(auto split: split-indicator intro!: $\Psi.\text{flowD-outside}[OF$
 $g]$ simp add: B-out)
also have $\dots = d\text{-OUT } g \langle x \rangle$ **using that** **by**(rule IN-g-B)
also have $(\sum^+ y. g (\langle y \rangle, \langle x \rangle) * \text{indicator } \mathbf{E} (y, x)) = d\text{-OUT } ?g x$ **unfolding**
d-OUT-def **using that**

```

    by(intro nn-integral-cong)(auto split: split-indicator intro!:  $\Psi$ .flowD-outside[OF
g] simp add: B-out)
    also have  $\dots = d\text{-IN } g \langle x \rangle - g \text{ (SOURCE, } \langle x \rangle)$  using that by(rule OUT-g-B)
    also have  $d\text{-IN } (h \ 0) \ x + d\text{-OUT } g \langle x \rangle - \dots = d\text{-IN } (h \ 0) \ x + g \text{ (SOURCE, } \langle x \rangle)$ 
    using  $\Psi$ .flowD-finite-IN[OF g] g-le
    by(cases d-IN (h 0) x; cases d-IN g  $\langle x \rangle$ ; cases d-IN g  $\langle x \rangle$ ; cases g (SOURCE,
 $\langle x \rangle$ ))
    (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  $\Gamma \ h'$ 
proof
  fix x
  consider (A)  $x \in A \ \Gamma \mid (B) \ x \in B \ \Gamma \mid (\text{outside}) \ x \notin \mathbf{V}$  using bipartite-V by
auto
  note cases = this

  show  $d\text{-OUT } h' \ x \leq \text{weight } \Gamma \ x$ 
  proof(cases rule: cases)
    case A
    then have  $d\text{-OUT } h' \ x = d\text{-OUT } (h \ 0) \ x + g \text{ (} \langle x \rangle, \text{SINK}) - g \text{ (SINK, } \langle x \rangle)$ 
  by(simp add: OUT-h'-A)
    also have  $\dots \leq d\text{-OUT } (h \ 0) \ x + g \text{ (} \langle x \rangle, \text{SINK})$  by(rule diff-le-self-ennreal)
    also have  $g \text{ (} \langle x \rangle, \text{SINK}) \leq \text{weight } \Gamma \ x - d\text{-OUT } (h \ 0) \ x$ 
    using flowD-capacity[OF g, of  $\langle x \rangle, \text{SINK}$ ] A by simp
    also have  $d\text{-OUT } (h \ 0) \ x + \dots = \text{weight } \Gamma \ x$ 
    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  $d\text{-IN } h' \ x \leq \text{weight } \Gamma \ x$ 
  proof(cases rule: cases)
    case B
    hence  $d\text{-IN } h' \ x = d\text{-IN } (h \ 0) \ x + g \text{ (SOURCE, } \langle x \rangle)$  by(rule IN-h'-B)
    also have  $\dots \leq \text{weight } \Gamma \ x$ 
    by(simp add: g-SOURCE  $\alpha$  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  $e \notin \mathbf{E}$  for e using that by(simp split: prod.split-asm add:
h'-sims)
qed
moreover
have SAT-h':  $B \ \Gamma \cap \mathbf{V} \subseteq \text{SAT } \Gamma \ h'$ 

```



```

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  $d-IN \ h' \ x = weight \ \Gamma \ x$  using that True
      by(simp add: IN-h'-B g-SOURCE  $\alpha$  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  $d-IN \ h' \ x = d-IN \ (h \ 0) \ x$  using that False by(simp add: IN-h'-B
g-SOURCE)
      also have  $\dots = weight \ \Gamma \ x$ 
      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  $(\exists z \in set \ p. z \in B \ \Gamma \cap \mathbf{V}) \vee x \in B \ \Gamma \cap \mathbf{V}$  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  $\Gamma \ (TER \ h')$  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-locals (auto 4 3 simp add: weight-outside )

```

lemma *unhinder:* — Lemma 6.9

```

  assumes loose: loose  $\Gamma$ 
  and  $b: b \in B \ \Gamma$ 
  and  $wb: weight \ \Gamma \ b > 0$ 

```

and $\delta: \delta > 0$
shows $\exists \varepsilon > 0. \varepsilon < \delta \wedge \neg \text{hindered} (\text{reduce-weight } \Gamma \ b \ \varepsilon)$
proof(*rule ccontr*)
assume $\neg ?thesis$
hence *hindered*: *hindered* (*reduce-weight* $\Gamma \ b \ \varepsilon$) **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** $wb = \text{enn2real} (\text{weight } \Gamma \ b)$
have *wb-conv*: $\text{weight } \Gamma \ b = \text{ennreal } wb$ **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** $n :: \text{nat}$
have $\varepsilon\text{-pos}$: $\varepsilon \ n > 0$ **for** n **using** *wb-pos* δ **by**(*simp add: \varepsilon-def*)
have $\varepsilon\text{-nonneg}$: $0 \leq \varepsilon \ n$ **for** n **using** $\varepsilon\text{-pos}[of \ n]$ **by** *simp*
have $*$: $\varepsilon \ n \leq \min wb \ \delta / 2$ **for** n **using** *wb-pos* δ
by(*auto simp add: \varepsilon-def field-simps min-def*)
have $\varepsilon\text{-le}$: $\varepsilon \ n \leq wb$ **and** $\varepsilon\text{-less}$: $\varepsilon \ n < wb$ **and** $\varepsilon\text{-less-}\delta$: $\varepsilon \ n < \delta$ **and** $\varepsilon\text{-le'}$: $\varepsilon \ n \leq wb / 2$ **for** n
using $*[of \ n]$ $\varepsilon\text{-pos}[of \ n]$ **by**(*auto*)

define Γ' **where** $\Gamma' \ n = \text{reduce-weight } \Gamma \ b \ (\varepsilon \ n)$ **for** $n :: \text{nat}$
have $\Gamma'\text{-sel}$ [*simp*]:
 $\text{edge } (\Gamma' \ n) = \text{edge } \Gamma$
 $A \ (\Gamma' \ n) = A \ \Gamma$
 $B \ (\Gamma' \ n) = B \ \Gamma$
 $\text{weight } (\Gamma' \ n) \ x = \text{weight } \Gamma \ x - (\text{if } x = b \text{ then } \varepsilon \ n \text{ else } 0)$
 $\text{essential } (\Gamma' \ n) = \text{essential } \Gamma$
 $\text{roofed-gen } (\Gamma' \ n) = \text{roofed-gen } \Gamma$
for $n \ x$ **by**(*simp-all add: \Gamma'-def*)

have $\text{vertex-}\Gamma'$ [*simp*]: $\text{vertex } (\Gamma' \ n) = \text{vertex } \Gamma$ **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* $\Gamma' \ n$ **for** n **unfolding** $\Gamma'\text{-def}$
using *wb-pos* **by**(*intro countable-bipartite-web-reduce-weight*)(*simp-all add: wb-conv \varepsilon-le \varepsilon-nonneg*)

obtain g **where** $g: \bigwedge n. \text{current } (\Gamma' \ n) \ (g \ n)$
and $w: \bigwedge n. \text{wave } (\Gamma' \ n) \ (g \ n)$
and hind : $\bigwedge n. \text{hindrance } (\Gamma' \ n) \ (g \ n)$ **using** *hindered[OF \varepsilon-pos, unfolded wb-conv ennreal-less-iff, OF \varepsilon-less-}\delta]*
unfolding *hindered.simps* $\Gamma'\text{-def}$ **by** *atomize-elim metis*
from g **have** $g\Gamma$: $\text{current } \Gamma \ (g \ n)$ **for** n
by(*rule current-weight-mono*)(*auto simp add: \varepsilon-nonneg diff-le-self-ennreal*)
note [*simp*] = *currentD-finite[OF g\Gamma]*

have $b\text{-TER}$: $b \in \text{TER}_{\Gamma' n} (g n)$ **for** n
proof(*rule ccontr*)
assume b' : $b \notin \text{TER}_{\Gamma' n} (g n)$
then have TER : $\text{TER}_{\Gamma' n} (g n) = \text{TER} (g n)$ **using** $b \varepsilon\text{-nonneg}$ [*of n*]
by(*auto simp add: SAT.simps split: if-split-asm intro: ennreal-diff-le-mono-left*)
from w [*of n*] TER **have** $\text{wave } \Gamma (g n)$ **by**(*simp add: wave.simps separating-gen.simps*)
moreover have $\text{hindrance } \Gamma (g n)$ **using** hind [*of n*] $\text{TER } bnA b'$
by(*auto simp add: hindrance.simps split: if-split-asm*)
ultimately show False **using** loose-unhindered [*OF loose*] $g\Gamma$ [*of n*] **by**(*auto intro: hindered.intros*)
qed

have $\text{IN-}g\text{-}b$: $d\text{-IN} (g n) b = \text{weight } \Gamma b - \varepsilon n$ **for** n **using** $b\text{-TER}$ [*of n*] bnA
by(*auto simp add: currentD-SAT[OF g]*)

define factor where $\text{factor } n = (wb - \varepsilon 0) / (wb - \varepsilon n)$ **for** n
have factor-le-1: $\text{factor } n \leq 1$ **for** n **using** $wb\text{-pos } \delta \varepsilon\text{-less}$ [*of n*]
by(*auto simp add: factor-def field-simps $\varepsilon\text{-def min-def}$*)
have factor-pos: $0 < \text{factor } n$ **for** n **using** $wb\text{-pos } \delta * \varepsilon\text{-less}$ **by**(*simp add: factor-def field-simps*)
have factor: $(wb - \varepsilon n) * \text{factor } n = wb - \varepsilon 0$ **for** n **using** $\varepsilon\text{-less}$ [*of n*]
by(*simp add: factor-def field-simps*)

define g' **where** $g' = (\lambda n (x, y). \text{if } y = b \text{ then } g n (x, y) * \text{factor } n \text{ else } g n (x, y))$
have $g'\text{-simps}$: $g' n (x, y) = (\text{if } y = b \text{ then } g n (x, y) * \text{factor } n \text{ else } g n (x, y))$
for $n x y$ **by**(*simp add: g'-def*)
have $g'\text{-le-}g$: $g' n e \leq g n e$ **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 $4 + (n * 6 + n * (n * 2)) \neq (0 :: \text{real})$ **for** $n :: \text{nat}$
by(*metis (mono-tags, opaque-lifting) add-is-0 of-nat-eq-0-iff of-nat-numeral zero-neq-numeral*)

then have $\text{IN-}g'$: $d\text{-IN} (g' n) x = (\text{if } x = b \text{ then } \text{weight } \Gamma b - \varepsilon 0 \text{ else } d\text{-IN} (g n) x)$ **for** $x n$
using $b\text{-TER}$ [*of n*] bnA factor-pos [*of n*] factor [*of n*] $wb\text{-pos } \delta$
by(*auto simp add: d-IN-def g'-simps nn-integral-divide nn-integral-cmult currentD-SAT[OF g] wb-conv $\varepsilon\text{-def field-simps}$ ennreal-minus ennreal-mult'[symmetric] intro!: arg-cong[where $f = \text{ennreal}$]*)
have $\text{OUT-}g'$: $d\text{-OUT} (g' n) x = d\text{-OUT} (g n) x - g n (x, b) * (1 - \text{factor } n)$
for $n x$
proof –
have $d\text{-OUT} (g' n) x = (\sum^+ y. g n (x, y)) - (\sum^+ y. (g n (x, y) * (1 - \text{factor } n))) * \text{indicator } \{b\} y$
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 $\dots = d\text{-OUT } (g \ n) \ x - g \ n \ (x, b) * (1 - \text{factor } n)$ **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 d-OUT $(g' \ n) \ x \leq \text{weight } (\Gamma' \ 0) \ x$ **for** x
using b-TER[of n] currentD-weight-OUT[OF g, of n x] $\varepsilon\text{-le}[of \ 0]$ factor-le-1[of
n]
by(auto simp add: OUT-g' SINK.simps ennreal-diff-le-mono-left)
show d-IN $(g' \ n) \ x \leq \text{weight } (\Gamma' \ 0) \ x$ **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 $\varepsilon\text{-def}$)
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': SINK $(g \ n) = \text{SINK } (g' \ n)$ **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': SAT $(\Gamma' \ n) \ (g \ n) = \text{SAT } (\Gamma' \ 0) \ (g' \ n)$ **for** n **using** b-TER[of n]
 $\varepsilon\text{-le}'[of \ 0]$
by(auto simp add: SAT.simps wb-conv IN-g' IN-g-b)
have TER-g': $\text{TER}_{\Gamma' \ n} (g \ n) = \text{TER}_{\Gamma' \ 0} (g' \ n)$ **for** n
using b-TER[of n] **by**(auto simp add: SAT.simps SINK-g' OUT-g' IN-g'
wb-conv $\varepsilon\text{-def}$)

have w': wave $(\Gamma' \ 0) \ (g' \ n)$ **for** n
proof
have separating $(\Gamma' \ 0) \ (\text{TER}_{\Gamma' \ n} (g \ n))$ **using** waveD-separating[OF w, of n]
by(simp add: separating-gen.simps)
then show separating $(\Gamma' \ 0) \ (\text{TER}_{\Gamma' \ 0} (g' \ n))$ **unfolding** TER-g' .
qed(rule g')

define f **where** $f = \text{rec-nat } (g \ 0) \ (\lambda n \text{ rec. rec } \frown_{\Gamma' \ 0} g' \ (n + 1))$
have f-simps [simp]:
 $f \ 0 = g \ 0$
 $f \ (\text{Suc } n) = f \ n \ \frown_{\Gamma' \ 0} g' \ (n + 1)$
for n **by**(simp-all add: f-def)

```

have f: current  $(\Gamma' 0) (f n)$  and fw: wave  $(\Gamma' 0) (f n)$  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:  $n \leq m \implies f n \leq f m$  for n m
proof(induction m rule: dec-induct)
  case (step k)
  note step.IH
  also have  $f k \leq (f k \frown_{\Gamma' 0} g' (k + 1))$ 
  by(rule le-funI plus-web-greater)+
  also have  $\dots = f (Suc k)$  by simp
  finally show ?case .
qed simp
have chain-f: Complete-Partial-Order.chain  $(\leq)$  (range f)
  by(rule chain-imageI[where le-a= $(\leq)$ ])(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:  $RF (TER_{\Gamma' 0} (f n)) = RF (\bigcup_{i \leq n}. TER_{\Gamma' 0} (g' i))$  for n
proof(induction n)
  case 0 show ?case by(simp add: TER-g')
next
  case (Suc n)
  have  $RF (TER_{\Gamma' 0} (f (Suc n))) = RF_{\Gamma' 0} (TER_{\Gamma' 0} (f n \frown_{\Gamma' 0} g' (n + 1)))$ 
  by simp
  also have  $\dots = RF_{\Gamma' 0} (TER_{\Gamma' 0} (f n) \cup TER_{\Gamma' 0} (g' (n + 1)))$  using f fw
  g' w'
  by(rule RF-TER-plus-web)
  also have  $\dots = RF_{\Gamma' 0} (RF_{\Gamma' 0} (TER_{\Gamma' 0} (f n)) \cup TER_{\Gamma' 0} (g' (n + 1)))$ 
  by(simp add: roofed-idem-Un1)
  also have  $RF_{\Gamma' 0} (TER_{\Gamma' 0} (f n)) = RF_{\Gamma' 0} (\bigcup_{i \leq n}. TER_{\Gamma' 0} (g' i))$  by(simp
  add: Suc.IH)
  also have  $RF_{\Gamma' 0} (\dots \cup TER_{\Gamma' 0} (g' (n + 1))) = RF_{\Gamma' 0} ((\bigcup_{i \leq n}. TER_{\Gamma' 0}
  (g' i)) \cup TER_{\Gamma' 0} (g' (n + 1)))$ 
  by(simp add: roofed-idem-Un1)
  also have  $(\bigcup_{i \leq n}. TER_{\Gamma' 0} (g' i)) \cup TER_{\Gamma' 0} (g' (n + 1)) = (\bigcup_{i \leq Suc n}.
  TER_{\Gamma' 0} (g' i))$ 
  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 $SINK$ - $g\omega$: $SINK \ g\omega = (\bigcap n. SINK \ (f \ n))$ **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 ω*)

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' : $edge \ (quotient\text{-}web \ (\Gamma' \ 0) \ (g' \ n)) = edge \ (quotient\text{-}web \ (\Gamma' \ n) \ (g \ n))$ **for** n
by(*simp add: fun-eq-iff TER-g' RF-circ*)
have $restrict$ -curr- g' : $f \upharpoonright \Gamma' \ 0 \ / \ g' \ n = f \upharpoonright \Gamma' \ n \ / \ g \ n$ **for** $n \ f$
by(*simp add: restrict-current-def RF-circ TER-g'*)

have RF -restrict: $roofed\text{-}gen \ (quotient\text{-}web \ (\Gamma' \ n) \ (g \ n)) = roofed\text{-}gen \ (quotient\text{-}web \ (\Gamma' \ 0) \ (g' \ n))$ **for** n
by(*simp add: roofed-def fun-eq-iff edge-restrict- Γ'*)

have $g\omega r'$: *current* $(quotient\text{-}web \ (\Gamma' \ 0) \ (g' \ n)) \ (g\omega \upharpoonright \Gamma' \ 0 \ / \ g' \ n)$ **for** n **using** $w' \ g\omega$
by(*rule current-restrict-current*)
have $g\omega r$: *current* $(quotient\text{-}web \ (\Gamma' \ n) \ (g \ n)) \ (g\omega \upharpoonright \Gamma' \ n \ / \ g \ n)$ **for** n **using** $w \ g\omega'$
by(*rule current-restrict-current*)
have $w\omega r$: *wave* $(quotient\text{-}web \ (\Gamma' \ n) \ (g \ n)) \ (g\omega \upharpoonright \Gamma' \ n \ / \ g \ n)$ (**is** *wave* ? Γ' ? $g\omega$)
for n
proof
have *: *wave* $(quotient\text{-}web \ (\Gamma' \ 0) \ (g' \ n)) \ (g\omega \upharpoonright \Gamma' \ 0 \ / \ g' \ n)$
using $g' \ w' \ g\omega \ w\omega$ **by**(*rule wave-restrict-current*)
have d -IN $(g\omega \upharpoonright \Gamma' \ n \ / \ g \ n) \ b = 0$
by(*rule d-IN-restrict-current-outside roofed-greaterI b-TER*)+
hence SAT -subset: $SAT \ (quotient\text{-}web \ (\Gamma' \ 0) \ (g' \ n)) \ (g\omega \upharpoonright \Gamma' \ n \ / \ g \ n) \subseteq SAT$
 $? \Gamma' \ (g\omega \upharpoonright \Gamma' \ n \ / \ g \ n)$
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 $TER\text{-}subset: TER_{quotient\text{-}web}(\Gamma' \ 0) \ (g' \ n) \ (g\omega \upharpoonright \Gamma' \ n \ / \ g \ n) \subseteq TER_{\Gamma'}$
 $(g\omega \upharpoonright \Gamma' \ n \ / \ g \ n)$
using $SINK\text{-}g'$ **by** $(auto \ simp \ add: \ restrict\text{-}curr\text{-}g')$

show $separating \ ?\Gamma' \ (TER_{\Gamma'} \ ?g\omega) \ (\text{is separating} - \ ?TER)$
proof
fix $x \ y \ p$
assume $xy: x \in A \ ?\Gamma' \ y \in B \ ?\Gamma' \ \text{and} \ p: path \ ?\Gamma' \ x \ p \ y$
from p **have** $p': path \ (quotient\text{-}web \ (\Gamma' \ 0) \ (g' \ n)) \ x \ p \ y$ **by** $(simp \ add: \ edge\text{-}restrict\text{-}\Gamma')$
with $waveD\text{-}separating[OF \ *, \ THEN \ separatingD, \ simplified, \ OF \ p]$ $TER\text{-}g'[of \ n]$ $SINK\text{-}g' \ SAT\text{-}g' \ restrict\text{-}curr\text{-}g' \ SAT\text{-}subset \ xy$
show $(\exists z \in set \ p. \ z \in \ ?TER) \vee x \in \ ?TER$ **by** $auto$
qed

show $d\text{-}OUT \ (g\omega \upharpoonright \Gamma' \ n \ / \ g \ n) \ x = 0$ **if** $x \notin RF_{\Gamma'} \ ?TER$ **for** x
unfolding $restrict\text{-}curr\text{-}g'[symmetric]$ **using** $TER\text{-}subset$ **that**
by $(intro \ waveD\text{-}OUT[OF \ *])(auto \ simp \ add: \ TER\text{-}g' \ restrict\text{-}curr\text{-}g' \ RF\text{-}restrict \ intro: \ in\text{-}roofed\text{-}mono)$
qed

have $RF\text{-}g\omega: RF \ (TER_{\Gamma'} \ 0 \ g\omega) = RF \ (\bigcup n. \ TER_{\Gamma'} \ 0 \ (g' \ n))$
proof $-$
have $RF_{\Gamma'} \ 0 \ (TER_{\Gamma'} \ 0 \ g\omega) = RF \ (\bigcup i. \ TER_{\Gamma'} \ 0 \ (f \ i))$
unfolding $g\omega\text{-}def$ **by** $(subst \ RF\text{-}TER\text{-}Sup[OF \ - \ - \ chain\text{-}f])(auto \ simp \ add: \ f \ fw \ supp\text{-}f)$
also have $\dots = RF \ (\bigcup i. \ RF \ (TER_{\Gamma'} \ 0 \ (f \ i)))$ **by** $(simp \ add: \ roofed\text{-}UN)$
also have $\dots = RF \ (\bigcup i. \ \bigcup j \leq i. \ TER_{\Gamma'} \ 0 \ (g' \ j))$ **unfolding** $RF\text{-}f \ roofed\text{-}UN$
by $simp$
also have $(\bigcup i. \ \bigcup j \leq i. \ TER_{\Gamma'} \ 0 \ (g' \ j)) = (\bigcup i. \ TER_{\Gamma'} \ 0 \ (g' \ i))$ **by** $auto$
finally show $?thesis$ **by** $simp$
qed

have $SAT\text{-}plus\text{-}\omega: SAT \ (\Gamma' \ n) \ (g \ n \ \frown_{\Gamma' \ n} \ g\omega) = SAT \ (\Gamma' \ 0) \ (g' \ n \ \frown_{\Gamma' \ 0} \ g\omega)$
for n
apply $(intro \ set\text{-}eqI)$
apply $(simp \ add: \ SAT.simps \ IN\text{-}plus\text{-}current[OF \ g \ w \ g\omega r] \ IN\text{-}plus\text{-}current[OF \ g' \ w' \ g\omega r']) \ TER\text{-}g')$
apply $(cases \ d\text{-}IN \ (g\omega \upharpoonright \Gamma' \ n \ / \ g \ n) \ b)$
apply $(auto \ simp \ add: \ SAT.simps \ wb\text{-}conv \ d\text{-}IN\text{-}plus\text{-}web \ IN\text{-}g')$
apply $(simp\text{-}all \ add: \ wb\text{-}conv \ IN\text{-}g\text{-}b \ restrict\text{-}curr\text{-}g' \ \varepsilon\text{-}def \ field\text{-}simps)$
apply $(metis \ TER\text{-}g' \ b\text{-}TER \ roofed\text{-}greaterI) +$
done
have $SINK\text{-}plus\text{-}\omega: SINK \ (g \ n \ \frown_{\Gamma' \ n} \ g\omega) = SINK \ (g' \ n \ \frown_{\Gamma' \ 0} \ g\omega)$ **for** n
apply $(rule \ set\text{-}eqI; \ simp \ add: \ SINK.simps \ OUT\text{-}plus\text{-}current[OF \ g \ w \ g\omega r] \ OUT\text{-}plus\text{-}current[OF \ g' \ w' \ g\omega r'] \ current\text{-}restrict\text{-}current[OF \ w' \ g\omega])$
using $factor\text{-}pos[of \ n]$
by $(auto \ simp \ add: \ RF\text{-}circ \ TER\text{-}g' \ restrict\text{-}curr\text{-}g' \ currentD\text{-}OUT\text{-}eq\text{-}0[OF \ g])$

```

currentD-OUT-eq-0[OF g'] g'-simps split: if-split-asm)
have TER-plus- $\omega$ :  $TER_{\Gamma' n} (g n \frown_{\Gamma' n} g\omega) = TER_{\Gamma' 0} (g' n \frown_{\Gamma' 0} g\omega)$  for  $n$ 
  by(rule set-eqI iffI)+(simp-all add: SAT-plus- $\omega$  SINK-plus- $\omega$ )

define h where h n = g n  $\frown_{\Gamma' n}$  g $\omega$  for n
have h: current ( $\Gamma' n$ ) (h n) for n unfolding h-def using g w
  by(rule current-plus-current)(rule current-restrict-current[OF w g $\omega$ '])
have hw: wave ( $\Gamma' n$ ) (h n) for n unfolding h-def using g w g $\omega'$  w $\omega$ r 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  $RF_{\Gamma' 0} (TER_{\Gamma' n} (h n)) = RF_{\Gamma' 0} (RF_{\Gamma' 0} (TER_{\Gamma' 0} g\omega) \cup TER_{\Gamma' 0} (g' n))$ 
  unfolding h-def TER-plus- $\omega$  RF-TER-plus-web[OF g' w' g $\omega$  w $\omega$ ] roofed-idem-Un1
  by(simp add: Un-commute)
  also have ... = RF ( $(\bigcup n. TER_{\Gamma' 0} (g' n)) \cup TER_{\Gamma' 0} (g' n)$ )
  by(simp add: RF-g $\omega$  roofed-idem-Un1)
  also have ... =  $RF_{\Gamma' 0} T$  unfolding T-def
  by(auto simp add: RF-g $\omega$  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-g $\omega$ [THEN
in-SINK-restrict-current]
  by(auto simp add: h-def OUT-plus-current[OF g w g $\omega$ r] SINK.simps)
have OUT-h- $\mathcal{E}$ : d-OUT (h n) x = 0 if x  $\in \mathcal{E}$  T for x n using that
  apply(subst (asm)  $\mathcal{E}$ -RF[symmetric])
  apply(subst (asm) (1 2) RF-h[symmetric, of n])
  apply(subst (asm)  $\mathcal{E}$ -RF)
  apply(simp add: SINK.simps)
done
have IN-h- $\mathcal{E}$ : d-IN (h n) x = weight ( $\Gamma' n$ ) x if x  $\in \mathcal{E}$  T x  $\notin A \Gamma$  for x n using
that
  apply(subst (asm)  $\mathcal{E}$ -RF[symmetric])
  apply(subst (asm) (1 2) RF-h[symmetric, of n])
  apply(subst (asm)  $\mathcal{E}$ -RF)
  apply(simp add: currentD-SAT[OF h])
done

have b-SAT: b  $\in SAT (\Gamma' 0) (h 0)$  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-g $\omega$  b-TER by(simp add: T-def)(metis SAT-g $\omega$ 
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  $p: \text{path } \Gamma \ b \ p \ y$  and  $y: y \in B \ \Gamma$  and distinct: distinct
    ( $b \neq p$ )
    and bypass:  $\bigwedge z. z \in \text{set } p \implies z \notin RF \ (TER_{\Gamma', 0} (h \ 0))$  by(rule  $\mathcal{E}$ -E-RF)
  auto
  from bypass have bypass':  $\bigwedge z. z \in \text{set } p \implies z \notin T$  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  $h0: \text{current } \Gamma \ (h \ 0)$  using  $h[\text{of } 0]$  by(rule current-weight-mono)(simp-all
add: wb-conv  $\varepsilon$ -nonneg)
moreover have  $\text{wave } \Gamma \ (h \ 0)$ 
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  $\varepsilon\text{-nonneg}[\text{of}$ 
 $0] \ b$ 
  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  $h \ 0 = \text{zero-current}$  by(rule looseD-wave[OF loose])
then have d-IN  $(h \ 0) \ b = 0$  by(simp)
with b-SAT  $\text{wb } \langle b \notin A \ \Gamma \rangle$  show False by(simp add: SAT.simps wb-conv  $\varepsilon$ -def
ennreal-minus-if split: if-split-asm)
qed

define S where  $S = \{x \in RF \ (T \cap B \ \Gamma) \cap A \ \Gamma. \text{essential } \Gamma \ (T \cap B \ \Gamma) \ (RF$ 
 $(T \cap B \ \Gamma) \cap A \ \Gamma) \ x\}$ 
define  $\Gamma\text{-}h$  where  $\Gamma\text{-}h = \langle \text{edge} = \lambda x \ y. \text{edge } \Gamma \ x \ y \wedge x \in S \wedge y \in T \wedge y \in B$ 
 $\Gamma, \text{weight} = \lambda x. \text{weight } \Gamma \ x * \text{indicator } (S \cup T \cap B \ \Gamma) \ x, A = S, B = T \cap B \ \Gamma \rangle$ 
have  $\Gamma\text{-}h\text{-sel}$  [simp]:
   $\text{edge } \Gamma\text{-}h \ x \ y \iff \text{edge } \Gamma \ x \ y \wedge x \in S \wedge y \in T \wedge y \in B \ \Gamma$ 
   $A \ \Gamma\text{-}h = S$ 
   $B \ \Gamma\text{-}h = T \cap B \ \Gamma$ 
   $\text{weight } \Gamma\text{-}h \ x = \text{weight } \Gamma \ x * \text{indicator } (S \cup T \cap B \ \Gamma) \ x$ 
for  $x \ y$ 
by(simp-all add:  $\Gamma\text{-}h\text{-def}$ )

have vertex- $\Gamma$ -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  $\Gamma$ -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  $\Gamma$ 
  (  $T \cap B \Gamma$  ) (  $RF (T \cap B \Gamma) \cap A \Gamma$  )  $x$ 
  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:  $\bigwedge z. z \in set \ p \implies z \notin RF (T \cap B \Gamma) \cap A \Gamma$  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 ( $A$ )  $x \in A \Gamma$  | ( $B$ )  $x \in B \Gamma$  | (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

```

$add: rtraccl-path-simps)$
with $y' \in T$ **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** $essential\ \Gamma\ (T \cap B\ \Gamma)\ (RF\ (T \cap B\ \Gamma) \cap A\ \Gamma)\ x$
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 $B\text{-vertex}: vertex\ \Gamma\text{-}h\ y$ **if** $T: y \in T$ **and** $B: y \in B\ \Gamma$ **and** $w: weight\ \Gamma\ y > 0$ **for** y
proof –
from $T\ B$ **disjoint** $\varepsilon\text{-less}[of\ 0]\ w$
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 $\Gamma\text{-}h: countable\text{-bipartite}\text{-web}\ \Gamma\text{-}h$
proof
show $V_{\Gamma\text{-}h} \subseteq A\ \Gamma\text{-}h \cup B\ \Gamma\text{-}h$ **by**($auto$ $simp$ $add: vertex-def$)
show $A\ \Gamma\text{-}h \subseteq V_{\Gamma\text{-}h}$ **using** $S\text{-vertex}$ **by** $auto$
show $x \in A\ \Gamma\text{-}h \wedge y \in B\ \Gamma\text{-}h$ **if** $edge\ \Gamma\text{-}h\ x\ y$ **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 $E_{\Gamma\text{-}h} \subseteq E$ **by** $auto$
thus $countable\ E_{\Gamma\text{-}h}$ **by**($rule\ countable\ subset$) $simp$
show $weight\ \Gamma\text{-}h\ x \neq \top$ **for** x **by**($simp$ $split: split-indicator$)
show $weight\ \Gamma\text{-}h\ x = 0$ **if** $x \notin V_{\Gamma\text{-}h}$ **for** x
using $that\ S\text{-vertex}\ B\text{-vertex}[of\ x]$
by($cases\ weight\ \Gamma\text{-}h\ x > 0$)($auto$ $split: split-indicator$)
qed
then **interpret** $\Gamma\text{-}h: countable\text{-bipartite}\text{-web}\ \Gamma\text{-}h$.

have $essential\text{-}T: essential\ \Gamma\ (B\ \Gamma)\ T = essential\ \Gamma\ (B\ \Gamma)\ (TER_{\Gamma'}\ 0\ (h\ 0))$
proof($rule\ ext\ iffI$)+
fix x
assume $essential\ \Gamma\ (B\ \Gamma)\ T\ x$
then **obtain** $p\ y$ **where** $p: path\ \Gamma\ x\ p\ y$ **and** $y: y \in B\ \Gamma$ **and** $distinct: distinct$

```

(x # p)
  and bypass:  $\bigwedge z. z \in \text{set } p \implies z \notin RF \ T$  by(rule essentialE-RF)auto
  from bypass have bypass':  $\bigwedge z. z \in \text{set } p \implies z \notin TER_{\Gamma', 0} (h \ 0)$ 
  unfolding RF-h[of 0, symmetric] by(blast intro: roofed-greaterI)
  show essential  $\Gamma \ (B \ \Gamma) \ (TER_{\Gamma', 0} (h \ 0)) \ x$  using p y
  by(blast intro: essentialI dest: bypass')
next
  fix x
  assume essential  $\Gamma \ (B \ \Gamma) \ (TER_{\Gamma', 0} (h \ 0)) \ x$ 
  then obtain p y where p: path  $\Gamma \ x \ p \ y$  and y:  $y \in B \ \Gamma$  and distinct: distinct
(x # p)
  and bypass:  $\bigwedge z. z \in \text{set } p \implies z \notin RF \ (TER_{\Gamma', 0} (h \ 0))$  by(rule essentialE-RF)auto
  from bypass have bypass':  $\bigwedge z. z \in \text{set } p \implies z \notin T$ 
  unfolding RF-h[of 0] by(blast intro: roofed-greaterI)
  show essential  $\Gamma \ (B \ \Gamma) \ T \ x$  using p y
  by(blast intro: essentialI dest: bypass')
qed

have h': current  $\Gamma$ -h (h n) for n
proof
  show d-OUT (h n)  $x \leq \text{weight } \Gamma$ -h x for x
  using currentD-weight-OUT[OF h, of n x]  $\varepsilon$ -nonneg[of n]  $\Gamma'$ .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)  $x \leq \text{weight } \Gamma$ -h x for x
  using currentD-weight-IN[OF h, of n x] currentD-IN[OF h, of x n]  $\varepsilon$ -nonneg[of
n] b-T b  $\Gamma'$ .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  $e \notin \mathbf{E}_{\Gamma-h}$  for e
  using that OUT-not-S[of fst e n] currentD-outside'[OF h, of e n]  $\Gamma'$ .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] cur-
rentD-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':  $B \ \Gamma$ -h  $\cap \mathbf{V}_{\Gamma-h} - \{b\} \subseteq SAT \ \Gamma$ -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** $\text{vertex}: x \in \mathbf{V}_{\Gamma\text{-}h}$ **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 $d\text{-}IN (h n) x = \text{weight } (\Gamma' n) x$ **using** xnA **by**(*rule IN-h-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 $(\lambda n. \min \delta \text{ } wb * (1 / (\text{real } (n + 2)))) \longrightarrow 0$
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 $(INF n. \text{ennreal } (\varepsilon n)) = 0$ **using** $wb\text{-pos } \delta$
apply(*simp add: \varepsilon-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 $(SUP n. d\text{-}IN (h n) b) = \text{weight } \Gamma\text{-}h b$ **using** $\text{essential } b \text{ } bnA \text{ } wb$
 $IN\text{-}h\text{-}\mathcal{E}[of b]$
by(*simp add: SUP-const-minus-ennreal*)
moreover have $d\text{-}IN (h 0) b \leq d\text{-}IN (h n) b$ **for** n **using** $\text{essential } b \text{ } bnA \text{ } wb\text{-pos } \delta$
 $IN\text{-}h\text{-}\mathcal{E}[of b]$
by(*simp add: wb-conv \varepsilon-def field-simps ennreal-minus-if min-le-iff-disj*)
moreover have $b\text{-}V: b \in \mathbf{V}_{\Gamma\text{-}h}$ **using** b *wb essential* **by**(*auto dest: B-vertex*)
ultimately have $\exists h'. \text{current } \Gamma\text{-}h h' \wedge \text{wave } \Gamma\text{-}h h' \wedge B \Gamma\text{-}h \cap \mathbf{V}_{\Gamma\text{-}h} \subseteq SAT$
 $\Gamma\text{-}h h'$
by(*rule \Gamma\text{-}h.unhinder-bipartite[OF h']*)
then obtain h' **where** $h': \text{current } \Gamma\text{-}h h'$ **and** $h'w: \text{wave } \Gamma\text{-}h h'$
and $B\text{-}SAT': B \Gamma\text{-}h \cap \mathbf{V}_{\Gamma\text{-}h} \subseteq SAT \Gamma\text{-}h h'$ **by** *blast*

have $h'': \text{current } \Gamma h'$
proof
show $d\text{-}OUT h' x \leq \text{weight } \Gamma x$ **for** x **using** *currentD-weight-OUT[OF h', of x]*
by(*auto split: split-indicator-asm elim: order-trans intro:*)
show $d\text{-}IN h' x \leq \text{weight } \Gamma x$ **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 $\text{wave } \Gamma h'$
proof
have *separating* $(\Gamma' 0) T$ **unfolding** $T\text{-def}$ **by**(*rule waveD-separating[OF ww]*)
hence *separating* ΓT **by**(*simp add: separating-gen.simps*)

```

hence *: separating  $\Gamma$  ( $\mathcal{E}$   $T$ ) by(rule separating-essential)
show separating  $\Gamma$  ( $TER$   $h'$ )
proof
  fix  $x$   $p$   $y$ 
  assume  $x$ :  $x \in A$   $\Gamma$  and  $p$ : 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)
  from separatingD[OF *  $p$   $x$   $y$ ]  $py$  have  $x \in \mathcal{E}$   $T \vee y \in \mathcal{E}$   $T$  by auto
  then show  $(\exists z \in set\ p. z \in TER\ h') \vee x \in TER\ h'$ 
  proof cases
    case left
    then have  $x \notin V_{\Gamma-h}$  using  $x$  disjoint
    by(auto 4 4 dest!: vertex- $\Gamma$ -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  $x \in TER\ h'$  by(auto simp add: SAT.A SINK.simps)
    thus ?thesis ..
  next
  case right
  have  $y \in SAT\ \Gamma\ h'$ 
  proof(cases weight  $\Gamma$   $y > 0$ )
    case True
    with  $py$   $x$   $y$  right have vertex  $\Gamma$ -h  $y$  by(auto intro: B-vertex)
    hence  $y \in SAT\ \Gamma$ -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  $y \in TER\ h'$  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  $b \in SAT\ \Gamma$ -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*: $\text{web-flow } \Gamma f \longleftrightarrow \text{current } \Gamma f$
using *bipartite-V* **by** (*auto simp add: web-flow.simps*)

lemma *countable-bipartite-web-minus-web*:

assumes f : *current* Γf

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* Γf

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* $?fg x \leq \text{weight } \Gamma x$ **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* $?fg x \leq \text{weight } \Gamma x$ **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* Γf

and w : *wave* Γf

```

and  $g$ : current  $(\Gamma \ominus f)$   $g$ 
and  $w'$ : wave  $(\Gamma \ominus f)$   $g$ 
shows wave  $\Gamma$  (plus-current  $f$   $g$ ) (is wave -  $?fg$ )
proof
  show  $fg$ : current  $\Gamma$   $?fg$  using  $f$   $g$  by(rule current-plus-current-minus)
  show  $sep$ : separating  $\Gamma$  (TER  $?fg$ )
  proof
    fix  $x$   $p$   $y$ 
    assume  $x$ :  $x \in A$   $\Gamma$  and  $p$ : 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)
    with waveD-separating[THEN separatingD, OF w p x y] have  $x \in TER$   $f \vee y$ 
     $\in TER$   $f$  by auto
    thus  $(\exists z \in set\ p. z \in TER\ ?fg) \vee x \in TER\ ?fg$ 
    proof cases
      case right
        with  $y$  disjoint have  $y \in TER$   $?fg$  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  $(\Gamma \ominus f)$   $x$   $p$   $y$  by simp
          from waveD-separating[THEN separatingD, OF w' this]  $x$   $y$   $py$ 
          have  $x \in TER_{\Gamma \ominus f}$   $g \vee y \in TER_{\Gamma \ominus f}$   $g$  by auto
          thus  $?thesis$ 
          proof cases
            case left
              hence  $x \in TER$   $?fg$  using  $x$   $x'$ 
              by(auto simp add: SAT.simps SINK.simps d-OUT-def nn-integral-add)
              thus  $?thesis$  ..
            next
              case right
                hence  $y \in TER$   $?fg$  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
      qed
    qed

lemma minus-plus-current:
  assumes  $f$ : current  $\Gamma$   $f$ 
  and  $g$ : current  $(\Gamma \ominus f)$   $g$ 
  shows  $\Gamma \ominus plus-current\ f\ g = \Gamma \ominus f \ominus g$  (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: $\bigwedge z. z \in \text{set } p \implies z \notin RF(TER ?fg)$* **by**(*rule $\mathcal{E}$ -E-RF*) *blast*
from *p a y disjoint* **have** *py: p = [y]*
by(*cases*)(*auto 4 3 elim: rtranc-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** $y \notin SAT \Gamma ?fg$ **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  $\Gamma$ 
  and f: current  $\Gamma$  f
  and w: wave  $\Gamma$  f
  and maximal:  $\bigwedge w. \llbracket \text{current } \Gamma w; \text{wave } \Gamma w; f \leq w \rrbracket \implies f = w$ 
  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  $\Gamma$  ?g by (rule current-plus-current-minus)
  moreover from f w g w' have wave  $\Gamma$  ?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  $f \ e \leq d\text{-OUT } f \ x$  unfolding d-OUT-def Pair by (rule nn-integral-ge-point)
  simp
    also have  $\dots \leq \text{weight } \Gamma \ x$  by (rule currentD-weight-OUT [OF f])
    also have  $\dots < \top$  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  $g = (\lambda \cdot. 0)$  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  $x < \text{weight } ? \Gamma \ x$  by cases
    have hindrance  $\Gamma \ f$ 
    proof
      show a':  $x \in A \ \Gamma$  using a by simp
      with weight show d-OUT  $f \ x < \text{weight } \Gamma \ x$ 
      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  $\Gamma \ x \ p \ y$  and y:  $y \in B \ \Gamma$ 
      and bypass:  $\bigwedge z. z \in \text{set } p \implies z \notin RF \ (TER \ f)$  by (rule  $\mathcal{E}\text{-E-RF}$ ) 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
    qed
  qed

```

```

      by(auto simp add: SINK.simps SAT.simps diff-gr0-ennreal)
    hence  $y \notin \text{TER}_{\Gamma} \text{ zero-current}$  using  $y$  disjoint by(auto)
    hence essential ?T (B ?T) (TERΓ zero-current)  $x$  using  $p \ y \ py$  by(auto
intro!: essentialI)
    with  $a$  have  $x \in \mathcal{E}_{\Gamma}$  (TERΓ zero-current) by simp
    with  $\mathcal{E}$  show False by contradiction
  qed
qed
thus hindered  $\Gamma$  using  $f \ w \ ..$ 
qed
qed

```

```

lemma weight-minus-web:
  assumes  $f$ : current  $\Gamma \ f$ 
  shows  $\text{weight} (\Gamma \ominus f) \ x = (\text{if } x \in A \ \Gamma \text{ then } \text{weight} \ \Gamma \ x - d\text{-OUT } f \ x \text{ else } \text{weight} \ \Gamma \ x - d\text{-IN } f \ x)$ 
proof(cases  $x \in B \ \Gamma$ )
  case True
    with currentD-OUT[OF  $f$  True] disjoint show ?thesis by auto
  next
    case False
    hence  $d\text{-IN } f \ x = 0 \ d\text{-OUT } f \ x = 0$  if  $x \notin A \ \Gamma$ 
      using currentD-outside-OUT[OF  $f$ , of  $x$ ] currentD-outside-IN[OF  $f$ , of  $x$ ] bi-
partite- $V$  that by auto
    then show ?thesis by simp
qed

```

```

lemma (in  $-$ ) separating-minus-web [simp]: separating-gen  $(G \ominus f) = \text{separating-gen } G$ 
by(simp add: separating-gen.simps fun-eq-iff)

```

```

lemma current-minus:
  assumes  $f$ : current  $\Gamma \ f$ 
  and  $g$ : current  $\Gamma \ g$ 
  and  $le$ :  $\bigwedge e. g \ e \leq f \ e$ 
  shows current  $(\Gamma \ominus g) \ (f - g)$ 
proof -
  interpret  $\Gamma$ : 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  $d\text{-OUT } (f - g) \ x \leq \text{weight} (\Gamma \ominus g) \ x$  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  $d\text{-IN } (f - g) \ x \leq \text{weight} (\Gamma \ominus g) \ x$  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 $(f - g) \cdot e = 0$ **if** $e \notin E_{\Gamma \ominus g}$ **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 : $\bigwedge e. g \cdot e \leq f \cdot e$
shows wave-minus: wave $(\Gamma \ominus g) (f - g)$
and TER-minus: $TER\ f \subseteq TER_{\Gamma \ominus g} (f - g)$
proof
have $x \in SINK\ f \implies x \in SINK\ (f - g)$ **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 $x \in SAT\ \Gamma\ f \implies x \in SAT\ (\Gamma \ominus g) (f - g)$ **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 $TER\ f \subseteq TER_{\Gamma \ominus g} (f - g)$ **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 $d-OUT\ f\ x = 0$ **by**(rule waveD-OUT[OF w])
moreover have $0 \leq f \cdot e$ **for** e **using** le [of e] **by** simp
ultimately show $d-OUT\ (f - g)\ x = 0$ **unfolding** d-OUT-def
by(simp add: nn-integral-0-iff emeasure-count-space-eq-0)
qed

lemma (in $-$) essential-minus-web [simp]: essential $(\Gamma \ominus f) = essential\ \Gamma$
by(simp add: essential-def fun-eq-iff)

lemma (in $-$) RF-in-essential: fixes B **shows** essential $\Gamma\ B\ S\ x \implies x \in roofed-gen\ \Gamma\ B\ S \iff x \in S$
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 $d-OUT\ (f((x, y) := k))\ x' = (if\ x = x'\ then\ d-OUT\ f\ x - f(x, y) + k\ else\ d-OUT\ f\ x')$
(is ?lhs = ?rhs)
proof(cases $x = x'$)

```

case True
have ?lhs = ( $\sum^+ y'. f(x, y') + k * \text{indicator } \{y\} y'$ ) - ( $\sum^+ y'. f(x, y') * \text{indicator } \{y\} y'$ )
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 ( $\sum^+ y'. f(x, y') + k * \text{indicator } \{y\} y'$ ) = d-OUT f x + ( $\sum^+ y'. k * \text{indicator } \{y\} y'$ )
unfolding d-OUT-def using assms
by(subst nn-integral-add[symmetric])
  (auto intro!: nn-integral-cong split: split-indicator)
also have ... - ( $\sum^+ y'. f(x, y') * \text{indicator } \{y\} y'$ ) = ?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:  $\neg \text{hindered } \Gamma$ 
and a:  $a \in A \Gamma$ 
shows  $\exists f. \text{current } \Gamma f \wedge \text{d-OUT } f a = \text{weight } \Gamma a \wedge \neg \text{hindered } (\Gamma \oplus f)$ 
proof -
from a A-vertex have a-vertex:  $\text{vertex } \Gamma a$  by auto
from unhindered have  $\neg \text{hindrance } \Gamma \text{ zero-current}$  by(auto intro!: hindered.intros
simp add: )
then have a- $\mathcal{E}$ :  $a \in \mathcal{E}(A \Gamma)$  if  $\text{weight } \Gamma a > 0$ 
proof(rule contrapos-np)
assume  $a \notin \mathcal{E}(A \Gamma)$ 
with a have  $\neg \text{essential } \Gamma (B \Gamma) (A \Gamma) a$  by simp
hence  $\neg \text{essential } \Gamma (B \Gamma) (A \Gamma \cup \{x. \text{weight } \Gamma x \leq 0\}) a$ 
by(rule contrapos-nn)(erule essential-mono; simp)
with a that show  $\text{hindrance } \Gamma \text{ zero-current}$  by(auto intro: hindrance)
qed

```

```

define F where  $F = (\lambda(\varepsilon, h :: 'v \text{ current}). \text{plus-current } \varepsilon h)$ 
have F-simps:  $F(\varepsilon, h) = \text{plus-current } \varepsilon h$  for  $\varepsilon h$  by(simp add: F-def)
define Fld where  $Fld = \{(\varepsilon, h).$ 
   $\text{current } \Gamma \varepsilon \wedge (\forall x. x \neq a \longrightarrow \text{d-OUT } \varepsilon x = 0) \wedge$ 
   $\text{current } (\Gamma \oplus \varepsilon) h \wedge \text{wave } (\Gamma \oplus \varepsilon) h \wedge$ 
   $\neg \text{hindered } (\Gamma \oplus F(\varepsilon, h))\}$ 
define leq where  $\text{leq} = \text{restrict-rel } Fld \{(f, f'). f \leq f'\}$ 
have Fld:  $\text{Field } \text{leq} = Fld$  by(auto simp add: leq-def)
have F-I [intro?]:  $(\varepsilon, h) \in \text{Field } \text{leq}$ 
if  $\text{current } \Gamma \varepsilon$  and  $\bigwedge x. x \neq a \implies \text{d-OUT } \varepsilon x = 0$ 
and  $\text{current } (\Gamma \oplus \varepsilon) h$  and  $\text{wave } (\Gamma \oplus \varepsilon) h$ 
and  $\neg \text{hindered } (\Gamma \oplus F(\varepsilon, h))$ 
for  $\varepsilon h$  using that by(simp add: Fld Fld-def)
have  $\varepsilon\text{-curr}$ :  $\text{current } \Gamma \varepsilon$  if  $(\varepsilon, h) \in \text{Field } \text{leq}$  for  $\varepsilon h$  using that by(simp add:
Fld Fld-def)

```

have $OUT\text{-}\varepsilon$: $\bigwedge x. x \neq a \implies d\text{-}OUT \ \varepsilon \ x = 0$ **if** $(\varepsilon, h) \in Field \text{ leq}$ **for** $\varepsilon \ h$ **using** *that* **by**(*simp add: Fld Fld-def*)
have h : *current* $(\Gamma \ominus \varepsilon) \ h$ **if** $(\varepsilon, h) \in Field \text{ leq}$ **for** $\varepsilon \ h$ **using** *that* **by**(*simp add: Fld Fld-def*)
have $h\text{-}w$: *wave* $(\Gamma \ominus \varepsilon) \ h$ **if** $(\varepsilon, h) \in Field \text{ leq}$ **for** $\varepsilon \ h$ **using** *that* **by**(*simp add: Fld Fld-def*)
have $unhindered'$: $\neg hindered \ (\Gamma \ominus F \ \varepsilon h)$ **if** $\varepsilon h \in Field \text{ leq}$ **for** εh **using** *that* **by**(*simp add: Fld Fld-def split: prod.split-asm*)
have f : *current* $\Gamma \ (F \ \varepsilon h)$ **if** $\varepsilon h \in Field \text{ leq}$ **for** εh **using** $\varepsilon\text{-curr} \ h$ *that* **by**(*cases* εh)(*simp add: F-simps current-plus-current-minus*)
have $out\text{-}\varepsilon$: $\varepsilon \ (x, y) = 0$ **if** $(\varepsilon, h) \in Field \text{ leq}$ $x \neq a$ **for** $\varepsilon \ h \ x \ y$ **proof**(*rule antisym*)
have $\varepsilon \ (x, y) \leq d\text{-}OUT \ \varepsilon \ x$ **unfolding** $d\text{-}OUT\text{-}def$ **by**(*rule nn-integral-ge-point*)
simp
with $OUT\text{-}\varepsilon[OF \text{ that}]$ **show** $\varepsilon \ (x, y) \leq 0$ **by** *simp*
qed *simp*
have $IN\text{-}\varepsilon$: $d\text{-}IN \ \varepsilon \ x = \varepsilon \ (a, x)$ **if** $(\varepsilon, h) \in Field \text{ leq}$ **for** $\varepsilon \ h \ x$ **proof**(*rule trans*)
show $d\text{-}IN \ \varepsilon \ x = (\sum^+ y. \varepsilon \ (y, x) * indicator \ \{a\} \ y)$ **unfolding** $d\text{-}IN\text{-}def$ **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** $\varepsilon \leq \varepsilon' \ h \leq h' \ (\varepsilon, h) \in Field \text{ leq} \ (\varepsilon', h') \in Field \text{ leq}$ **for** $\varepsilon \ h \ \varepsilon' \ h'$
using *that* **unfolding** Fld **by**(*simp add: leq-def in-restrict-rel-iff*)

have $chain\text{-}Field$: $Sup \ M \in Field \text{ leq}$ **if** M : $M \in Chains \text{ leq}$ **and** $nempty$: $M \neq \{\}$ **for** M
unfolding $Sup\text{-}prod\text{-}def$
proof
from $nempty$ **obtain** $\varepsilon \ h$ **where** $in\text{-}M$: $(\varepsilon, h) \in M$ **by** *auto*
with M **have** $Field$: $(\varepsilon, h) \in Field \text{ leq}$ **by**(*rule Chains-FieldD*)

from M **have** $chain$: $Complete\text{-}Partial\text{-}Order.chain \ (\lambda \varepsilon \ \varepsilon'. (\varepsilon, \varepsilon') \in leq) \ M$
by(*intro Chains-into-chain*) *simp*
hence $chain'$: $Complete\text{-}Partial\text{-}Order.chain \ (\leq) \ M$
by(*auto simp add: chain-def leq-def in-restrict-rel-iff*)
hence $chain1$: $Complete\text{-}Partial\text{-}Order.chain \ (\leq) \ (fst \ ' \ M)$
and $chain2$: $Complete\text{-}Partial\text{-}Order.chain \ (\leq) \ (snd \ ' \ M)$
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** $support\text{-}flow \ (Sup \ (fst \ ' \ M)) \subseteq \mathbf{E}$ **by**(*auto elim!: support-flow.cases intro: ccontr*)
hence $supp\text{-}flow1$: *countable* $(support\text{-}flow \ (Sup \ (fst \ ' \ M)))$ **by**(*rule countable-subset*) *simp*

show $SM1$: *current* $\Gamma \ (Sup \ (fst \ ' \ M))$

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 $\Gamma \ominus \text{Sup (fst ' M)}$*
using *SM1 by*(rule *countable-bipartite-web-minus-web*)

let *?h = Sup (snd ' M)*
have *outside2: ?h (x, y) = 0 if $\neg \text{edge } \Gamma \ x \ y$ 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 $\subseteq \mathbf{E}$ 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) $\in M$. d-OUT $\varepsilon \ x$) 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) $\in M$. d-OUT $\varepsilon \ a$ 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: ($\bigsqcup \varepsilon h \in M$. d-OUT (fst εh) x) $\leq$ weight $\Gamma \ x$ for x*
using *currentD-weight-OUT*[*OF SM1, of x*] *OUT1*[*of x*] **by**(simp *add: split-beta*)
have *OUT1-nonneg: 0 $\leq$ ($\bigsqcup \varepsilon h \in M$. 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) $\in M$. d-IN $\varepsilon \ x$) for x*
by (subst *d-IN-Sup* [*OF chain1 - supp-flow1*])
(simp-all add: nempty split-beta image-comp)
have *IN1-le: ($\bigsqcup \varepsilon h \in M$. d-IN (fst εh) x) $\leq$ weight $\Gamma \ x$ for x*
using *currentD-weight-IN*[*OF SM1, of x*] *IN1*[*of x*] **by**(simp *add: split-beta*)
have *IN1-nonneg: 0 $\leq$ ($\bigsqcup \varepsilon h \in M$. 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) $\in M$. $\varepsilon \ (a, x)$) for x*
unfolding *IN1 by*(rule *SUP-cong*[*OF refl*])(auto *dest!: Chains-FieldD*[*OF M*]
IN- ε)

have *directed: $\exists \varepsilon k'' \in M$. $F \ (\text{snd } \varepsilon k) + F \ (\text{fst } \varepsilon k') \leq F \ (\text{snd } \varepsilon k'') + F \ (\text{fst } \varepsilon k'')$*
if *mono: $\bigwedge f \ g. (\bigwedge z. f \ z \leq g \ z) \implies F \ f \leq F \ g \ \varepsilon k \in M \ \varepsilon k' \in M$*
for *$\varepsilon k \ \varepsilon k'$ and $F :: - \Rightarrow \text{ennreal}$*
using *chainD*[*OF chain that(2-3)*]

proof cases
case left
hence *snd $\varepsilon k \leq \text{snd } \varepsilon k'$ by*(simp *add: leq-def less-eq-prod-def in-restrict-rel-iff*)
hence *$F \ (\text{snd } \varepsilon k) + F \ (\text{fst } \varepsilon k') \leq F \ (\text{snd } \varepsilon k') + F \ (\text{fst } \varepsilon k')$*
by(intro *add-right-mono mono*)(clarsimp simp *add: le-fun-def*)
with that show *?thesis by blast*

```

next
  case right
  hence  $\text{fst } \varepsilon k' \leq \text{fst } \varepsilon k$  by(simp add: leq-def less-eq-prod-def in-restrict-rel-iff)
  hence  $F(\text{snd } \varepsilon k) + F(\text{fst } \varepsilon k') \leq F(\text{snd } \varepsilon k) + F(\text{fst } \varepsilon k)$ 
  by(intro add-left-mono mono)(clarsimp simp add: le-fun-def)
  with that show ?thesis by blast
qed
  have directed-OUT:  $\exists \varepsilon k'' \in M. d\text{-OUT}(\text{snd } \varepsilon k) x + d\text{-OUT}(\text{fst } \varepsilon k') x \leq$ 
 $d\text{-OUT}(\text{snd } \varepsilon k'') x + d\text{-OUT}(\text{fst } \varepsilon k'') x$ 
  if  $\varepsilon k \in M \ \varepsilon k' \in M$  for  $x \ \varepsilon k \ \varepsilon k'$  by(rule directed; rule d-OUT-mono that)
  have directed-IN:  $\exists \varepsilon k'' \in M. d\text{-IN}(\text{snd } \varepsilon k) x + d\text{-IN}(\text{fst } \varepsilon k') x \leq d\text{-IN}(\text{snd } \varepsilon k'')$ 
 $x + d\text{-IN}(\text{fst } \varepsilon k'') x$ 
  if  $\varepsilon k \in M \ \varepsilon k' \in M$  for  $x \ \varepsilon k \ \varepsilon k'$  by(rule directed; rule d-IN-mono that)

let ? $\Gamma = \Gamma \ominus \text{Sup}(\text{fst } \varepsilon M)$ 

have hM2: current ? $\Gamma$  h if  $\varepsilon h: (\varepsilon, h) \in M$  for  $\varepsilon h$ 
proof
  from  $\varepsilon h$  have Field:  $(\varepsilon, h) \in \text{Field leq}$  by(rule Chains-FieldD[OF M])
  then have H: current  $(\Gamma \ominus \varepsilon) h$  and  $\varepsilon\text{-curr}'$ : current  $\Gamma \ \varepsilon$  by(rule h  $\varepsilon\text{-curr}$ )+
  from  $\varepsilon\text{-curr}'$  interpret  $\Gamma$ : countable-bipartite-web  $\Gamma \ominus \varepsilon$  by(rule count-
able-bipartite-web-minus-web)

  fix x
  have  $d\text{-OUT } h x \leq d\text{-OUT } ?h x$  using  $\varepsilon h$  by(intro d-OUT-mono)(auto intro:
SUP-upper2)
  also have OUT:  $\dots = (\text{SUP } h \in \text{snd } \varepsilon M. d\text{-OUT } h x)$  using chain2 -
supp-flow2
  by(rule d-OUT-Sup)(simp-all add: nempty)
  also have  $\dots = \dots + (\text{SUP } \varepsilon \in \text{fst } \varepsilon M. d\text{-OUT } \varepsilon x) - (\text{SUP } \varepsilon \in \text{fst } \varepsilon M.$ 
 $d\text{-OUT } \varepsilon x)$ 
  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  $\dots = (\text{SUP } (\varepsilon, k) \in M. d\text{-OUT } k x + d\text{-OUT } \varepsilon x) - (\text{SUP } \varepsilon \in \text{fst } \varepsilon$ 
 $M. d\text{-OUT } \varepsilon x)$  unfolding split-def
  by (subst SUP-add-directed-ennreal[OF directed-OUT])
  (simp-all add: image-comp)
  also have  $(\text{SUP } (\varepsilon, k) \in M. d\text{-OUT } k x + d\text{-OUT } \varepsilon x) \leq \text{weight } \Gamma x$ 
  apply(clarsimp dest!: Chains-FieldD[OF M] intro!: SUP-least)
  subgoal premises that for  $\varepsilon h$ 
  using currentD-weight-OUT[OF h[OF that], of x] currentD-weight-OUT[OF
 $\varepsilon\text{-curr}$ [OF that], of x]
  countable-bipartite-web-minus-web[OF  $\varepsilon\text{-curr}$ , 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  $(\text{SUP } \varepsilon \in \text{fst } \varepsilon M. d\text{-OUT } \varepsilon x) = d\text{-OUT}(\text{Sup}(\text{fst } \varepsilon M)) x$  using

```


OUT1

by (*simp add: split-beta image-comp*)
 finally show $d\text{-OUT } h \ x \leq \text{weight } ?\Gamma \ x$
 using $\Gamma.\text{currentD-OUT}'[OF \ h[OF \ \text{Field}], \text{ of } x] \ \text{currentD-weight-IN}[OF \ SM1, \text{ of } x]$ by (*auto simp add: ennreal-minus-mono*)

 have $d\text{-IN } h \ x \leq d\text{-IN } ?h \ x$ using εh by (*intro d-IN-mono*) (*auto intro: SUP-upper2*)
 also have $IN: \dots = (SUP \ h \in \text{snd } 'M. \ d\text{-IN } h \ x)$ using *chain2 - supp-flow2*
 by (*rule d-IN-Sup*) (*simp-all add: nempty*)
 also have $\dots = \dots + (SUP \ \varepsilon \in \text{fst } 'M. \ d\text{-IN } \varepsilon \ x) - (SUP \ \varepsilon \in \text{fst } 'M. \ d\text{-IN } \varepsilon \ x)$
 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 $\dots = (SUP \ (\varepsilon, k) \in M. \ d\text{-IN } k \ x + d\text{-IN } \varepsilon \ x) - (SUP \ \varepsilon \in \text{fst } 'M. \ d\text{-IN } \varepsilon \ x)$ unfolding *split-def*
 by (*subst SUP-add-directed-ennreal* [*OF directed-IN*])
 (*simp-all add: image-comp*)
 also have $(SUP \ (\varepsilon, k) \in M. \ d\text{-IN } k \ x + d\text{-IN } \varepsilon \ x) \leq \text{weight } \Gamma \ x$
 apply (*clarsimp dest!: Chains-FieldD* [*OF M*] *intro!: SUP-least*)
 subgoal premises *that* for $\varepsilon \ h$
 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 \varepsilon-curr, THEN countable-bipartite-web.currentD-OUT', OF that h[OF that], where x=x*]
currentD-OUT' [*OF \varepsilon-curr, OF that, where x=x*] *currentD-IN* [*OF \varepsilon-curr, OF that, of x*] *currentD-weight-IN* [*OF \varepsilon-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 $(SUP \ \varepsilon \in \text{fst } 'M. \ d\text{-IN } \varepsilon \ x) = d\text{-IN } (\text{Sup } (\text{fst } 'M)) \ x$
 using *IN1* by (*simp add: split-beta image-comp*)
 finally show $d\text{-IN } h \ x \leq \text{weight } ?\Gamma \ x$
 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 $\text{snd } 'M \neq \{\}$ by *simp*
 from *chain2 this - supp-flow2* show *current: current* $?\Gamma \ ?h$
 by (*rule current-Sup*) (*clarify; rule hM2; simp*)

 have $wM: \text{wave } ?\Gamma \ h$ if $(\varepsilon, h) \in M$ for $\varepsilon \ h$
 proof

```

    let ?T' =  $\Gamma \ominus \varepsilon$ 
    have subset:  $TER_{?T'} h \subseteq TER_{?T} h$ 
    using currentD-OUT'[OF SM1] currentD-OUT'[OF  $\varepsilon$ -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 ?T' ( $TER_{?T'}$ 
h) by(rule waveD-separating)
      then show separating ?T ( $TER_{?T} h$ ) using subset by(auto intro: separating-
weakening)
        qed(rule hM2[OF that])
        show wave: wave ?T ?h using chain2  $\langle \text{snd} \text{ ' } M \neq \{\} \rangle$  - 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  $\varepsilon h$ ) ' 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  $\Gamma f$  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 zE:  $z \notin \mathcal{E}_{? \Omega} (TER_{? \Omega} g)$ 
      and OUT-z:  $d-OUT g z < \text{weight } ? \Omega z$  by cases auto
      define  $\delta$  where  $\delta = \text{weight } ? \Omega z - d-OUT g z$ 
      have  $\delta$ -pos:  $\delta > 0$  using OUT-z by(simp add:  $\delta$ -def diff-gr0-ennreal del:
minus-web-sel)
      then have  $\delta$ -finite[simp]:  $\delta \neq \top$  using z
      by(simp-all add:  $\delta$ -def)

```

```

have  $\exists (\varepsilon, h) \in M. d\text{-OUT } f a < d\text{-OUT } (\text{plus-current } \varepsilon h) a + \delta$ 
proof(rule ccontr)
  assume  $\neg ?thesis$ 
  hence greater:  $d\text{-OUT } (\text{plus-current } \varepsilon h) a + \delta \leq d\text{-OUT } f a$  if  $(\varepsilon, h) \in M$ 
for  $\varepsilon h$  using that by auto

  have chain'': Complete-Partial-Order.chain  $(\leq) ((\lambda(\varepsilon, h). \text{plus-current } \varepsilon h)$ 
    ‘  $M$ )
    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:  $\delta$ -pos)
  also have  $d\text{-OUT } f a + \delta = (\text{SUP } (\varepsilon, h) \in M. d\text{-OUT } (\text{plus-current } \varepsilon h) a)$ 
+  $\delta$ 
    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  $\dots \leq d\text{-OUT } f a$ 
    unfolding ennreal-SUP-add-left[symmetric, OF nempty]
  proof(rule SUP-least, clarify)
    show  $d\text{-OUT } (\text{plus-current } \varepsilon h) a + \delta \leq d\text{-OUT } f a$  if  $(\varepsilon, h) \in M$  for  $\varepsilon h$ 
    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  $\varepsilon h$  where  $hM: (\varepsilon, h) \in M$  and close:  $d\text{-OUT } f a < d\text{-OUT } (\text{plus-current } \varepsilon h) a + \delta$  by blast
  have Field:  $(\varepsilon, h) \in \text{Field leq}$  using hM by(rule Chains-FieldD)[OF M]
  then have  $\varepsilon$ : current  $\Gamma \varepsilon$ 
    and unhindered-h:  $\neg \text{hindered } (\Gamma \ominus F (\varepsilon, h))$ 
    and h-curr: current  $(\Gamma \ominus \varepsilon) h$ 
    and h-w: wave  $(\Gamma \ominus \varepsilon) h$ 
    and OUT- $\varepsilon$ :  $x \neq a \implies d\text{-OUT } \varepsilon x = 0$  for  $x$ 
    by(rule  $\varepsilon$ -curr OUT- $\varepsilon$  h h-w unhindered') +
  let  $? \varepsilon h = \text{plus-current } \varepsilon h$ 
    have  $\varepsilon h\text{-curr}$ : current  $\Gamma ? \varepsilon h$  using Field unfolding F-simps[symmetric]
by(rule f)

  interpret h: countable-bipartite-web  $\Gamma \ominus \varepsilon$  using  $\varepsilon$  by(rule countable-bipartite-web-minus-web)
  interpret  $\varepsilon h$ : countable-bipartite-web  $\Gamma \ominus ? \varepsilon h$  using  $\varepsilon h\text{-curr}$  by(rule countable-bipartite-web-minus-web)
  note [simp del] = minus-web-sel(2)
    and [simp] = weight-minus-web[OF  $\varepsilon h\text{-curr}$ ] weight-minus-web[OF SM1, simplified]

  define k where  $k e = \text{Sup } (\text{fst } ' M) e - \varepsilon e$  for  $e$ 
  have k-simps:  $k(x, y) = \text{Sup } (\text{fst } ' M)(x, y) - \varepsilon(x, y)$  for  $x y$  by(simp add:

```

k -def)
have k -alt: $k(x, y) = (\text{if } x = a \wedge \text{edge } \Gamma x y \text{ then } \text{Sup } (fst \text{ ' } M) (a, y) - \varepsilon (a, y) \text{ else } 0) \text{ for } x y$
by (cases $x = a$)
(auto simp add: k -simps out- ε [OF Field] currentD-outside [OF ε] im-
age-comp
intro!: SUP-eq-const [OF nempty] dest!: Chains-FieldD [OF M]
intro: currentD-outside [OF ε -curr] out- ε)
have OUT- k : $d\text{-OUT } k x = (\text{if } x = a \text{ then } d\text{-OUT } (\text{Sup } (fst \text{ ' } M)) a - d\text{-OUT } \varepsilon a \text{ else } 0) \text{ for } x$
proof –
have $d\text{-OUT } k x = (\text{if } x = a \text{ then } (\sum^+ y. \text{Sup } (fst \text{ ' } M) (a, y) - \varepsilon (a, y)) \text{ else } 0)$
using currentD-outside[OF SM1] currentD-outside[OF ε]
by(auto simp add: k -alt d-OUT-def intro!: nn-integral-cong)
also have $(\sum^+ y. \text{Sup } (fst \text{ ' } M) (a, y) - \varepsilon (a, y)) = d\text{-OUT } (\text{Sup } (fst \text{ ' } M)) a - d\text{-OUT } \varepsilon a$
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 : $d\text{-IN } k y = (\text{if } \text{edge } \Gamma a y \text{ then } \text{Sup } (fst \text{ ' } M) (a, y) - \varepsilon (a, y) \text{ else } 0) \text{ for } y$
proof –
have $d\text{-IN } k y = (\sum^+ x. (\text{if } \text{edge } \Gamma x y \text{ then } \text{Sup } (fst \text{ ' } M) (a, y) - \varepsilon (a, y) \text{ else } 0) * \text{indicator } \{a\} x)$
unfolding d-IN-def **by**(rule nn-integral-cong)(auto simp add: k -alt outgoing-def split: split-indicator)
also have $\dots = (\text{if } \text{edge } \Gamma a y \text{ then } \text{Sup } (fst \text{ ' } M) (a, y) - \varepsilon (a, y) \text{ else } 0)$
using hM
by(auto simp add: max-def intro!: SUP-upper2)
finally show ?thesis .
qed

have OUT- εh : $d\text{-OUT } ?\varepsilon h x = d\text{-OUT } \varepsilon x + d\text{-OUT } h x \text{ for } x$
unfolding plus-current-def **by**(rule d-OUT-add)+
have IN- εh : $d\text{-IN } ?\varepsilon h x = d\text{-IN } \varepsilon x + d\text{-IN } h x \text{ for } x$
unfolding plus-current-def **by**(rule d-IN-add)+

have OUT1-le': $d\text{-OUT } (\text{Sup } (fst \text{ ' } M)) x \leq \text{weight } \Gamma x \text{ for } x$
using OUT1-le[of x] **unfolding** OUT1 **by** (simp add: split-beta')

have k : current $(\Gamma \ominus ?\varepsilon h) k$
proof
fix x
show $d\text{-OUT } k x \leq \text{weight } (\Gamma \ominus ?\varepsilon h) x$
using a OUT1-na[of x] currentD-weight-OUT[OF hM2[OF hM], of x]
currentD-weight-IN[OF εh -curr, of x]

```

      currentD-weight-IN[OF  $\varepsilon$ , 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- $\varepsilon$  OUT- $\varepsilon h$  IN- $\varepsilon h$  currentD-OUT'[OF  $\varepsilon$ ]
    IN- $\varepsilon$ [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 *: ( $\bigsqcup_{xa \in M} \text{fst } xa \ (a, x) \leq d\text{-IN } (\text{Sup } (\text{fst}' M)) \ x$ 
    unfolding IN1 by (intro SUP-subset-mono) (auto simp: split-beta'
    d-IN-ge-point)
    also have ...  $\leq \text{weight } \Gamma \ x$ 
    using IN1-le[of  $x$ ] IN1 by (simp add: split-beta')
    finally show d-IN  $k \ x \leq \text{weight } (\Gamma \ominus ?\varepsilon h) \ x$ 
    using currentD-weight-IN[OF  $\varepsilon h$ -curr, of  $x$ ] currentD-weight-OUT[OF
     $\varepsilon h$ -curr, of  $x$ ]
      currentD-weight-IN[OF hM2[OF hM], of  $x$ ] IN- $\varepsilon$ [OF Field, of  $x$ ] *)
    apply(auto simp add: IN-k outgoing-def IN- $\varepsilon h$  IN- $\varepsilon$  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  $q = (\sum^+ y \in B \ (\Gamma \ominus ?\varepsilon h). \ d\text{-IN } k \ y - d\text{-OUT } k \ y)$ 
  have  $q\text{-alt}: q = (\sum^+ y \in - \ A \ (\Gamma \ominus ?\varepsilon h). \ d\text{-IN } k \ y - d\text{-OUT } k \ y)$  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  $q\text{-simps}: q = d\text{-OUT } (\text{Sup } (\text{fst}' M)) \ a - d\text{-OUT } \varepsilon \ a$ 
  proof -
    have  $q = (\sum^+ y. \ d\text{-IN } k \ y)$  using  $a$  IN1 OUT1 OUT1-na unfolding  $q\text{-alt}$ 
    by(auto simp add: nn-integral-count-space-indicator OUT-k IN- $\varepsilon$ [OF Field]
    OUT- $\varepsilon$  currentD-outside[OF  $\varepsilon$ ] outgoing-def no-loop A-in IN-k intro!: nn-integral-cong
    split: split-indicator)
    also have ...  $= d\text{-OUT } (\text{Sup } (\text{fst}' M)) \ a - d\text{-OUT } \varepsilon \ a$  using cur-
    rentD-finite-OUT[OF  $\varepsilon$ , of  $a$ ] hM currentD-outside[OF SM1] currentD-outside[OF
     $\varepsilon$ ]
    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  $q\text{-finite}: q \neq \top$  using currentD-finite-OUT[OF SM1, of  $a$ ]
  by(simp add:  $q\text{-simps}$ )
  have  $q\text{-nonneg}: 0 \leq q$  using hM by(auto simp add:  $q\text{-simps}$  intro!: d-OUT-mono

```

```

SUP-upper2)
  have q-less- $\delta$ :  $q < \delta$  using close
  unfolding q-simps  $\delta$ -def OUT- $\varepsilon h$  OUT-f
  proof -
    let ?F = d-OUT (Sup (fst'M)) a and ?S = d-OUT (Sup (snd'M)) a
    and ? $\varepsilon$  = d-OUT  $\varepsilon$  a and ?h = d-OUT h a and ?w = weight ( $\Gamma \ominus f$ ) z
  - d-OUT g z
    have ?F + ?h  $\leq$  ?F + ?S
    using hM by (auto intro!: add-mono d-OUT-mono SUP-upper2)
    also assume ?F + ?S < ? $\varepsilon$  + ?h + ?w
    finally have ?h + ?F < ?h + (?w + ? $\varepsilon$ )
    by (simp add: ac-simps)
    then show ?F - ? $\varepsilon$  < ?w
    using currentD-finite-OUT[OF  $\varepsilon$ , 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  $\varepsilon h$ .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  $\varepsilon h$ .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 ( $\Gamma \ominus \text{Sup } (fst \text{ ' } M)$ )  $h$  using  $hM$  by(rule  $hM2$ )

let  $?T = \Gamma \ominus ?\varepsilon h \ominus k$ 
interpret  $\Gamma$ : web  $?T$  using  $k$  by(rule  $\varepsilon h.\text{web-minus-web}$ )
note [simp] =  $\varepsilon h.\text{weight-minus-web}[OF\ k]\ h.\text{weight-minus-web}[OF\ h\text{-curr}]$ 
   $\text{weight-minus-web}[OF\ f\text{-curr}]\ SM1.\text{weight-minus-web}[OF\ h',\ \text{simplified}]$ 

interpret  $\Omega$ : countable-bipartite-web  $\Gamma \ominus f$  using  $f\text{-curr}$  by(rule countable-bipartite-web-minus-web)

have *:  $\Gamma \ominus f = \Gamma \ominus \text{Sup } (fst \text{ ' } M) \ominus \text{Sup } (snd \text{ ' } M)$  unfolding  $f\text{-def}$   $F\text{-simps}$ 
  using  $SM1.\text{current}$  by(rule minus-plus-current)
have  $OUT\text{-}\varepsilon k$ :  $d\text{-OUT } (\text{Sup } (fst \text{ ' } M))\ x = d\text{-OUT } \varepsilon\ x + d\text{-OUT } k\ x$  for  $x$ 
  using  $OUT1\text{ '[of } x]\ \text{currentD-finite-OUT}[OF\ \varepsilon]\ hM$ 
  by(auto simp add: OUT-k OUT- $\varepsilon$  add-diff-self-ennreal SUP-upper2)
have  $IN\text{-}\varepsilon k$ :  $d\text{-IN } (\text{Sup } (fst \text{ ' } M))\ x = d\text{-IN } \varepsilon\ x + d\text{-IN } k\ x$  for  $x$ 
  using  $IN1\text{ '[of } x]\ \text{currentD-finite-IN}[OF\ \varepsilon]\ \text{currentD-outside}[OF\ \varepsilon]\ \text{currentD-outside}[OF\ \varepsilon\text{-curr}]$ 
  by(auto simp add: IN-k IN- $\varepsilon$ [OF Field] add-diff-self-ennreal split-beta nempty image-comp
     $\text{dest!}: \text{Chains-FieldD}[OF\ M]\ \text{intro!}: \text{SUP-eq-const}\ \text{intro}: \text{SUP-upper2}[OF\ hM]$ )
have **:  $?T = \Gamma \ominus \text{Sup } (fst \text{ ' } M) \ominus h$ 
proof(rule web.equality)
  show weight  $?T = \text{weight } (\Gamma \ominus \text{Sup } (fst \text{ ' } M) \ominus h)$ 
  using  $OUT\text{-}\varepsilon k\ OUT\text{-}\varepsilon h\ \text{currentD-finite-OUT}[OF\ \varepsilon]\ IN\text{-}\varepsilon k\ IN\text{-}\varepsilon h\ \text{currentD-finite-IN}[OF\ \varepsilon]$ 
  by(auto simp add: diff-add-eq-diff-diff-swap-ennreal diff-diff-commute-ennreal)
qed simp-all
have  $g'\text{-alt}$ :  $g' = \text{plus-current } (\text{Sup } (snd \text{ ' } M))\ g - h$ 
  by(simp add: fun-eq-iff g'-simps add-diff-eq-ennreal add commute)

have current ( $\Gamma \ominus \text{Sup } (fst \text{ ' } M)$ ) ( $\text{plus-current } (\text{Sup } (snd \text{ ' } M))\ g$ ) using
current  $g$  unfolding *
  by(rule SM1.current-plus-current-minus)
hence  $g'$ : current  $?T\ g'$  unfolding * **  $g'\text{-alt}$  using  $hM2[OF\ hM]$ 
  by(rule SM1.current-minus)(auto intro!: add-increasing2 SUP-upper2 hM)

have wave ( $\Gamma \ominus \text{Sup } (fst \text{ ' } M)$ ) ( $\text{plus-current } (\text{Sup } (snd \text{ ' } M))\ g$ ) using current
wave  $g\ g\text{-w}$ 
  unfolding * by(rule SM1.wave-plus-current-minus)
then have  $g'\text{-w}$ : wave  $?T\ g'$  unfolding * **  $g'\text{-alt}$  using  $hM2[OF\ hM]$ 
  by(rule SM1.wave-minus)(auto intro!: add-increasing2 SUP-upper2 hM)

have hindrance-by  $?T\ g'\ q$ 
proof
  show  $z \in A\ ?T$  using  $z$  by simp
  show  $z \notin \mathcal{E}_{?T}\ (TER_{?T}\ g')$ 
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 ( $simp\text{-}all\ add: SINK.simps$ )
from  $ess$  obtain  $p\ y$  where  $p: path\ \Gamma\ z\ p\ y$  and  $y: y \in B\ \Gamma$ 
and  $bypass: \bigwedge z. z \in set\ p \implies z \notin RF\ (TER_{\Gamma} g')$  by ( $rule\ essentialE\text{-}RF$ )
auto
from  $y$  have  $y': y \notin A\ \Gamma$  using  $disjoint$  by  $blast$ 
from  $p\ z\ y$  obtain  $py: p = [y]$  and  $edge: edge\ \Gamma\ z\ y$  using  $disjoint$ 
by ( $cases$ ) ( $auto\ 4\ 3\ elim: rtrancl\text{-}path.cases\ dest: bipartite\text{-}E$ )
hence  $yRF: y \notin RF\ (TER_{\Gamma} g')$  using  $bypass[of\ y]$  by ( $auto$ )
with  $wave\text{-}not\text{-}RF\text{-}IN\text{-}zero[OF\ g'\ g'\text{-}w, of\ y]$  have  $IN\text{-}g'\text{-}y: d\text{-}IN\ g'\ y = 0$ 
by ( $auto\ intro: roofed\text{-}greaterI$ )
with  $yRF\ y\ y'$  have  $w\text{-}y: weight\ \Gamma\ y > 0$  using  $currentD\text{-}OUT[OF\ g',$ 
of  $y]$ 
by ( $auto\ simp\ add: RF\text{-}in\text{-}B\ currentD\text{-}SAT[OF\ g']\ SINK.simps\ zero\text{-}less\text{-}iff\text{-}neq\text{-}zero$ )
have  $y \notin SAT\ (\Gamma \ominus f)\ g$ 
proof
assume  $y \in SAT\ (\Gamma \ominus f)\ g$ 
with  $y\ disjoint$  have  $IN\text{-}g\text{-}y: d\text{-}IN\ g\ y = weight\ (\Gamma \ominus f)\ y$  by ( $auto\ simp$ 
add:  $currentD\text{-}SAT[OF\ g]$ )
have  $0 < weight\ \Gamma\ y - d\text{-}IN\ (\bigsqcup_{x \in M. fst\ x})\ y - d\text{-}IN\ h\ y$ 
using  $y'\ w\text{-}y\ unfolding\ **$  by  $auto$ 
have  $d\text{-}IN\ g'\ y > 0$ 
using  $y'\ w\text{-}y\ hM\ unfolding\ **$ 
apply ( $simp\ add: IN\text{-}g'\ IN\text{-}f\ IN\text{-}g\text{-}y\ diff\text{-}add\text{-}eq\text{-}diff\text{-}diff\text{-}swap\text{-}ennreal$ )
apply ( $subst\ add\text{-}diff\text{-}eq\text{-}ennreal$ )
apply ( $auto\ intro!: SUP\text{-}upper2\ d\text{-}IN\text{-}mono\ simp: diff\text{-}add\text{-}self\text{-}ennreal$ 
diff\text{-}gt\text{-}0\text{-}iff\text{-}gt\text{-}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  $essential\ \Gamma\ (B\ \Gamma)\ (TER_{\Gamma} \ominus f\ g)\ z$  by ( $auto\ intro:$ 
essentialI)
moreover with  $z\ waveD\text{-}separating[OF\ g\text{-}w, THEN\ separating\text{-}RF\text{-}A]$ 
have  $z \in \mathcal{E}_{\Omega} (TER_{\Omega} g)$ 
by ( $auto\ simp\ add: RF\text{-}in\text{-}essential$ )
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 ( $simp\ add: \delta\text{-}def\ OUT\text{-}f\ diff\text{-}add\text{-}eq\text{-}diff\text{-}diff\text{-}swap\text{-}ennreal$ )
apply ( $subst\ (5)\ diff\text{-}diff\text{-}commute\text{-}ennreal$ )
apply ( $rule\ ennreal\text{-}minus\text{-}mono[OF\ \text{-}\ order\text{-}refl]$ )
apply ( $auto\ simp\ add: ac\text{-}simps\ diff\text{-}add\text{-}eq\text{-}diff\text{-}diff\text{-}swap\text{-}ennreal[symmetric]$ 
add\text{-}diff\text{-}self\text{-}ennreal\ image\text{-}comp
intro!: ennreal\text{-}minus\text{-}mono[OF\ order\text{-}refl]\ SUP\text{-}upper2[OF\ hM]
d\text{-}OUT\text{-}mono)
done

```


then show $q\text{-}z: q < \text{weight } ?\Gamma z - d\text{-OUT } g' z$ **using** $q\text{-less-}\delta$ **by** *simp*
then show $d\text{-OUT } g' z < \text{weight } ?\Gamma z$ **using** $q\text{-nonneg } z$
by(*auto simp add: less-diff-eq-ennreal less-top[symmetric] ac-simps $\Gamma.\text{currentD-finite-OUT}[OF$
 $g']$
 intro: le-less-trans[rotated] add-increasing)
qed
then have *hindered-by: hindered-by* ($\Gamma \ominus ?\varepsilon h \ominus k$) q **using** $g' g'\text{-}w$ **by**(*rule*
hindered-by.intros)
then have *hindered* ($\Gamma \ominus ?\varepsilon h$) **using** $q\text{-finite}$ **unfolding** $q\text{-def}$ **by** $-(\text{rule}$
 $\varepsilon h.\text{hindered-reduce-current}[OF k])$
with *unhindered-h* **show** *False* **unfolding** $F\text{-simps}$ **by** *contradiction*
qed
qed*

define *sat* **where** *sat* =
 ($\lambda(\varepsilon, h).$
 let
 $f = F(\varepsilon, h);$
 $k = \text{SOME } k. \text{current } (\Gamma \ominus f) k \wedge \text{wave } (\Gamma \ominus f) k \wedge (\forall k'. \text{current } (\Gamma \ominus f)$
 $k' \wedge \text{wave } (\Gamma \ominus f) k' \wedge k \leq k' \longrightarrow k = k')$
 in
 if $d\text{-OUT } (\text{plus-current } f k) a < \text{weight } \Gamma a$ *then*
 let
 $\Omega = \Gamma \ominus f \ominus k;$
 $y = \text{SOME } y. y \in \mathbf{OUT}_{\Omega} a \wedge \text{weight } \Omega y > 0;$
 $\delta = \text{SOME } \delta. \delta > 0 \wedge \delta < \text{enn2real } (\min (\text{weight } \Omega a) (\text{weight } \Omega y)) \wedge$
 $\neg \text{hindered } (\text{reduce-weight } \Omega y \delta)$
 in
 ($\text{plus-current } \varepsilon (\text{zero-current}((a, y) := \delta)), \text{plus-current } h k$)
 else (ε, h)

have *zero: (zero-current, zero-current) $\in$ Field leq*
by(*rule F-I*)(*simp-all add: unhindered F-def*)

have *a-TER: $a \in \text{TER}_{\Gamma \ominus F \varepsilon h} k$*
if that: $\varepsilon h \in \text{Field leq}$
and $k: \text{current } (\Gamma \ominus F \varepsilon h) k$ **and** $k\text{-}w: \text{wave } (\Gamma \ominus F \varepsilon h) k$
and less: $d\text{-OUT } (\text{plus-current } (F \varepsilon h) k) a < \text{weight } \Gamma a$ **for** $\varepsilon h k$
proof(*rule ccontr*)
assume $\neg ?thesis$
hence $\mathcal{E}: a \notin \mathcal{E}_{\Gamma \ominus F \varepsilon h} (\text{TER}_{\Gamma \ominus F \varepsilon h} k)$ **by** *auto*
from that have $f: \text{current } \Gamma (F \varepsilon h)$ **and** *unhindered: $\neg \text{hindered } (\Gamma \ominus F \varepsilon h)$*
by(*cases εh ; simp add: f unhindered'; fail*)+

from less have $d\text{-OUT } k a < \text{weight } (\Gamma \ominus F \varepsilon h) a$ **using** $a \text{currentD-finite-OUT}[OF$
 $f, \text{of } a]$
by(*simp add: d-OUT-def nn-integral-add less-diff-eq-ennreal add commute*
less-top[symmetric])
with $\mathcal{E}$ **have** *hindrance* ($\Gamma \ominus F \varepsilon h$) k **by**(*rule hindrance*)(*simp add: a*)

then have hindered $(\Gamma \ominus F \varepsilon h)$ using $k \text{ } k\text{-}w \text{ } ..$
 with unhindered show False by contradiction
 qed

note minus-web-sel(2)[simp del]

let $?P\text{-}y = \lambda \varepsilon h \ k \ y. y \in \mathbf{OUT}_{\Gamma \ominus F \varepsilon h \ominus k} a \wedge \text{weight } (\Gamma \ominus F \varepsilon h \ominus k) \ y > 0$
 have $Ex\text{-}y: Ex \ (?P\text{-}y \ \varepsilon h \ k)$
 if that: $\varepsilon h \in \text{Field leq}$
 and k : current $(\Gamma \ominus F \varepsilon h) \ k$ and $k\text{-}w$: wave $(\Gamma \ominus F \varepsilon h) \ k$
 and less: $d\text{-}OUT \ (\text{plus-current } (F \varepsilon h) \ k) \ a < \text{weight } \Gamma \ a$ for $\varepsilon h \ k$
 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 $f[OF \text{ that}]$ by(rule countable-bipartite-web-minus-web)
 note [simp] = weight-minus-web[OF $f[OF \text{ that}]$] $\Gamma.\text{weight-minus-web}[OF \ k]$

have hindrance $? \Omega$ zero-current
 proof
 show $a \in A \ ? \Omega$ using a by simp
 show $a \notin \mathcal{E}_{? \Omega} \ (TER_{? \Omega} \text{ zero-current})$ (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: $\bigwedge z. z \in \text{set } p \implies z \notin RF_{? \Omega} \ ?TER$ by(rule $\mathcal{E}\text{-}E\text{-}RF$)(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 $d\text{-}OUT \ (\text{plus-current } (F \varepsilon h) \ k) \ x = d\text{-}OUT \ (F \varepsilon h) \ x + d\text{-}OUT \ k \ x$ for x
 by(simp add: d-OUT-def nn-integral-add)
 then show $d\text{-}OUT \text{ zero-current } a < \text{weight } ? \Omega \ a$ using less a-TER[OF that $k \text{ } k\text{-}w \text{ less}$] 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 $\neg \ \text{hindered } ? \Omega$ using unhindered'[OF that] $k \text{ } k\text{-}w$ by(rule $\Gamma.\text{unhindered-minus-web}$)
 ultimately show False by contradiction
 qed

have increasing: $\varepsilon h \leq \text{sat } \varepsilon h \wedge \text{sat } \varepsilon h \in \text{Field leq}$ if $\varepsilon h \in \text{Field leq}$ for εh

proof(*cases* εh)
case (*Pair* εh)
with that have that: $(\varepsilon, h) \in \text{Field leq}$ **by** *simp*
have f : *current* $\Gamma (F (\varepsilon, h))$ **and** *unhindered:* $\neg \text{hindered} (\Gamma \ominus F (\varepsilon, h))$
and ε : *current* $\Gamma \varepsilon$
and h : *current* $(\Gamma \ominus \varepsilon) h$ **and** $h\text{-}w$: *wave* $(\Gamma \ominus \varepsilon) h$ **and** $OUT\text{-}\varepsilon$: $x \neq a \implies$
 $d\text{-}OUT \varepsilon x = 0$ **for** x
using that **by**(*rule* $f \text{ unhindered}' \varepsilon\text{-curr } OUT\text{-}\varepsilon h h\text{-}w$) +
interpret Γ : *countable-bipartite-web* $\Gamma \ominus F (\varepsilon, h)$ **using** f **by**(*rule countable-bipartite-web-minus-web*)
note [*simp*] = *weight-minus-web*[$OF f$]

let $?P\text{-}k = \lambda k. \text{current} (\Gamma \ominus F (\varepsilon, h)) k \wedge \text{wave} (\Gamma \ominus F (\varepsilon, h)) k \wedge (\forall k'. \text{current} (\Gamma \ominus F (\varepsilon, h)) k' \wedge \text{wave} (\Gamma \ominus F (\varepsilon, h)) k' \wedge k \leq k' \longrightarrow k = k')$
define k **where** $k = Eps ?P\text{-}k$
have $Ex ?P\text{-}k$ **by**(*intro ex-maximal-wave*)(*simp-all*)
hence $?P\text{-}k$ k **unfolding** $k\text{-def}$ **by**(*rule someI-ex*)
hence k : *current* $(\Gamma \ominus F (\varepsilon, h)) k$ **and** $k\text{-}w$: *wave* $(\Gamma \ominus F (\varepsilon, h)) k$
and *maximal:* $\bigwedge k'. \llbracket \text{current} (\Gamma \ominus F (\varepsilon, h)) k'; \text{wave} (\Gamma \ominus F (\varepsilon, h)) k'; k \leq k' \rrbracket \implies k = k'$ **by** *blast+*
note [*simp*] = $\Gamma.\text{weight-minus-web}[OF k]$

let $?fk = \text{plus-current} (F (\varepsilon, h)) k$
have $IN\text{-}fk$: $d\text{-}IN ?fk x = d\text{-}IN (F (\varepsilon, h)) x + d\text{-}IN k x$ **for** x
by(*simp add: d-IN-def nn-integral-add*)
have $OUT\text{-}fk$: $d\text{-}OUT ?fk x = d\text{-}OUT (F (\varepsilon, h)) x + d\text{-}OUT k x$ **for** x
by(*simp add: d-OUT-def nn-integral-add*)
have fk : *current* $\Gamma ?fk$ **using** $f k$ **by**(*rule current-plus-current-minus*)

show *?thesis*
proof(*cases* $d\text{-}OUT ?fk a < \text{weight } \Gamma a$)
case *less:* *True*

define Ω **where** $\Omega = \Gamma \ominus F (\varepsilon, h) \ominus k$
have $B\text{-}\Omega$ [*simp*]: $B \Omega = B \Gamma$ **by**(*simp add: \Omega-def*)

have *loose:* *loose* Ω **unfolding** $\Omega\text{-def}$ **using** *unhindered* $k k\text{-}w$ *maximal* **by**(*rule \Gamma.loose-minus-web*)
interpret Ω : *countable-bipartite-web* Ω **using** k **unfolding** $\Omega\text{-def}$
by(*rule \Gamma.countable-bipartite-web-minus-web*)

have $a\text{-}\mathcal{E}$: $a \in \text{TER}_{\Gamma \ominus F (\varepsilon, h)} k$ **using** *that* $k k\text{-}w$ *less* **by**(*rule a-TER*)
then have $\text{weight-}\Omega\text{-}a$: $\text{weight } \Omega a = \text{weight } \Gamma a - d\text{-}OUT (F (\varepsilon, h)) a$
using a *disjoint* **by**(*auto simp add: roofed-circ-def \Omega-def SINK.simps*)
then have $\text{weight-}a$: $0 < \text{weight } \Omega a$ **using** *less* $a\text{-}\mathcal{E}$
by(*simp add: OUT-fk SINK.simps diff-gr0-ennreal*)

let $?P\text{-}y = \lambda y. y \in \text{OUT}_{\Omega} a \wedge \text{weight } \Omega y > 0$
define y **where** $y = Eps ?P\text{-}y$

have $Ex \text{ ?}P\text{-}y$ **using** *that k k-w less unfolding* $\Omega\text{-def}$ **by**(*rule Ex-y*)
hence $\text{?}P\text{-}y$ **unfolding** $y\text{-def}$ **by**(*rule someI-ex*)
hence $y\text{-OUT}$: $y \in \mathbf{OUT}_\Omega$ **and** $\text{weight-}y$: $\text{weight } \Omega \ y > 0$ **by** *blast+*
from $y\text{-OUT}$ **have** $y\text{-B}$: $y \in B \ \Omega$ **by**(*auto simp add: outgoing-def* $\Omega\text{-def}$ *dest: bipartite-E*)
with $\text{weight-}y$ **have** yRF : $y \notin RF_{\Gamma \ominus F(\varepsilon, h)} \ (TER_{\Gamma \ominus F(\varepsilon, h)} \ k)$
unfolding $\Omega\text{-def}$ **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 $IN\text{-}k\text{-}y$: $d\text{-IN } k \ y = 0$ **by**(*rule wave-not-RF-IN-zero*[$OF \ k \ k\text{-}w$])

define bound **where** $\text{bound} = \text{enn2real } (\min (\text{weight } \Omega \ a) (\text{weight } \Omega \ y))$
have bound-pos : $\text{bound} > 0$ **using** $\text{weight-}y$ $\text{weight-}a$ **using** Ω .*weight-finite*
by(*cases weight* $\Omega \ a$ *weight* $\Omega \ y$ *rule: ennreal2-cases*)
(simp-all add: bound-def min-def split: if-split-asm)

let $\text{?}P\text{-}\delta = \lambda \delta. \delta > 0 \wedge \delta < \text{bound} \wedge \neg \text{hindered } (\text{reduce-weight } \Omega \ y \ \delta)$
define δ **where** $\delta = \text{Eps } \text{?}P\text{-}\delta$
let $\text{?}\Omega = \text{reduce-weight } \Omega \ y \ \delta$

from Ω .*unhinder*[$OF \ \text{loose} - \text{weight-}y \ \text{bound-pos}$] $y\text{-B}$ *disjoint*
have $Ex \text{ ?}P\text{-}\delta$ **by**(*auto simp add:* $\Omega\text{-def}$)
hence $\text{?}P\text{-}\delta \ \delta$ **unfolding** $\delta\text{-def}$ **by**(*rule someI-ex*)
hence $\delta\text{-pos}$: $0 < \delta$ **and** $\delta\text{-le-bound}$: $\delta < \text{bound}$ **and** $\text{unhindered}'$: $\neg \text{hindered}$
 $\text{?}\Omega$ **by** *blast+*
from $\delta\text{-pos}$ **have** $\delta\text{-nonneg}$: $0 \leq \delta$ **by** *simp*
from $\delta\text{-le-bound}$ $\delta\text{-pos}$ **have** $\delta\text{-le-}a$: $\delta < \text{weight } \Omega \ a$ **and** $\delta\text{-le-}y$: $\delta < \text{weight } \Omega$
 y
by(*cases weight* $\Omega \ a$ *weight* $\Omega \ y$ *rule: ennreal2-cases*;
simp add: bound-def min-def ennreal-less-iff split: if-split-asm)

let $\text{?}\Gamma = \Gamma \ominus \text{?}fk$
interpret Γ' : *countable-bipartite-web* $\text{?}\Gamma$ **by**(*rule countable-bipartite-web-minus-web*
 fk)
note [*simp*] = *weight-minus-web*[$OF \ fk$]

let $\text{?}g = \text{zero-current}((a, y) := \delta)$
have $OUT\text{-}g$: $d\text{-OUT } \text{?}g \ x = (\text{if } x = a \text{ then } \delta \text{ else } 0)$ **for** x
proof(*rule trans*)
show $d\text{-OUT } \text{?}g \ x = (\sum^+ z. (\text{if } x = a \text{ then } \delta \text{ else } 0) * \text{indicator } \{y\} \ z)$
unfolding $d\text{-OUT-def}$
by(*rule nn-integral-cong*) *simp*
show $\dots = (\text{if } x = a \text{ then } \delta \text{ else } 0)$ **using** $\delta\text{-pos}$ **by**(*simp add: max-def*)
qed
have $IN\text{-}g$: $d\text{-IN } \text{?}g \ x = (\text{if } x = y \text{ then } \delta \text{ else } 0)$ **for** x
proof(*rule trans*)
show $d\text{-IN } \text{?}g \ x = (\sum^+ z. (\text{if } x = y \text{ then } \delta \text{ else } 0) * \text{indicator } \{a\} \ z)$
unfolding $d\text{-IN-def}$
by(*rule nn-integral-cong*) *simp*

```

    show ... = (if x = y then  $\delta$  else 0) using  $\delta$ -pos by(simp add: max-def)
qed

have g: current ?T ?g
proof
  show d-OUT ?g x  $\leq$  weight ?T 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-enreal-def[symmetric])
    next
      case True
        then show ?thesis using  $\delta$ -le-a a a- $\mathcal{E}$   $\delta$ -pos unfolding OUT-g
          by(simp add: OUT-g  $\Omega$ -def SINK.simps OUT-fk split: if-split-asm)
    qed
  show d-IN ?g x  $\leq$  weight ?T 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-enreal-def[symmetric])
    next
      case True
        then show ?thesis using  $\delta$ -le-y y-B a- $\mathcal{E}$   $\delta$ -pos currentD-OUT[OF k, of y] IN-k-y
          by(simp add: OUT-g  $\Omega$ -def SINK.simps OUT-fk IN-fk IN-g split: if-split-asm)
    qed
  show ?g e = 0 if e  $\notin$   $\mathbf{E}_T$  for e using y-OUT that by(auto simp add:  $\Omega$ -def outgoing-def)
  qed
  interpret  $\Gamma''$ : web  $\Gamma \ominus ?fk \ominus ?g$  using g by(rule  $\Gamma'$ .web-minus-web)

  let ? $\varepsilon'$  = plus-current  $\varepsilon$  (zero-current((a, y) :=  $\delta$ ))
  let ?h' = plus-current h k
  have F': F (? $\varepsilon'$ , ?h') = plus-current (plus-current (F ( $\varepsilon$ , h)) k) (zero-current((a, y) :=  $\delta$ )) (is - = ?f')
    by(auto simp add: F-simps fun-eq-iff add-ac)
  have sat: sat ( $\varepsilon$ , h) = (? $\varepsilon'$ , ?h') using less
    by(simp add: sat-def k-def  $\Omega$ -def Let-def y-def bound-def  $\delta$ -def)

  have le: ( $\varepsilon$ , h)  $\leq$  (? $\varepsilon'$ , ?h') using  $\delta$ -pos
    by(auto simp add: le-fun-def add-increasing2 add-increasing)

  have current ( $\Gamma \ominus \varepsilon$ ) (( $\lambda$ -. 0)((a, y) := enreal  $\delta$ )) using g
    by(rule current-weight-mono)(auto simp add: weight-minus-web[OF  $\varepsilon$ ] intro!: enreal-minus-mono d-OUT-mono d-IN-mono, simp-all add: F-def add-increasing2)
  with  $\varepsilon$  have  $\varepsilon'$ : current  $\Gamma$  ? $\varepsilon'$  by(rule current-plus-current-minus)

```

moreover have $eq-0: d-OUT \text{ ?}\varepsilon' x = 0$ **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 $\Gamma \ominus \text{ ?}\varepsilon'$ **by**(rule countable-bipartite-web-minus-web)
from ε **interpret** ε : countable-bipartite-web $\Gamma \ominus \varepsilon$ **by**(rule countable-bipartite-web-minus-web)
have g' : current $(\Gamma \ominus \varepsilon)$ $\text{ ?}g$ **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 $(\Gamma \ominus \varepsilon \ominus h)$ k **using** k **unfolding** F -simps minus-plus-current[OF
 εh].
with h **have** current $(\Gamma \ominus \varepsilon)$ (plus-current $h k$) **by**(rule ε .current-plus-current-minus)
hence current $(\Gamma \ominus \varepsilon)$ (plus-current (plus-current $h k$) $\text{ ?}g$) **using** g **unfolding**
minus-plus-current[OF $f k$]
unfolding F -simps minus-plus-current[OF εh] ε .minus-plus-current[OF h
 k' , symmetric]
by(rule ε .current-plus-current-minus)
then have current $(\Gamma \ominus \varepsilon \ominus \text{ ?}g)$ (plus-current (plus-current $h k$) $\text{ ?}g - \text{ ?}g$)
using g'
by(rule ε .current-minus)(auto simp add: add-increasing)
then have h'' : current $(\Gamma \ominus \text{ ?}\varepsilon')$ $\text{ ?}h'$
by(rule arg-cong2[**where** f =current, THEN iffD1, rotated -1])
(simp-all add: minus-plus-current[OF $\varepsilon g'$] fun-eq-iff add-diff-eq-ennreal[symmetric])
moreover have wave $(\Gamma \ominus \text{ ?}\varepsilon')$ $\text{ ?}h'$
proof
have separating $(\Gamma \ominus \varepsilon)$ ($TER_{\Gamma \ominus \varepsilon}$ (plus-current $h k$))
using k $k-w$ **unfolding** F -simps minus-plus-current[OF εh]
by(intro waveD-separating ε .wave-plus-current-minus[OF $h h-w$])
moreover have $TER_{\Gamma \ominus \varepsilon}$ (plus-current $h k$) $\subseteq TER_{\Gamma \ominus \text{ ?}\varepsilon'}$ (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 $(\Gamma \ominus \text{ ?}\varepsilon')$ ($TER_{\Gamma \ominus \text{ ?}\varepsilon'}$ $\text{ ?}h'$)
by(simp add: minus-plus-current[OF $\varepsilon g'$] separating-weakening)
qed(rule h'')
moreover
have $\neg$ hindered $(\Gamma \ominus F (\text{ ?}\varepsilon', \text{ ?}h'))$ **using** unhindered'
proof(rule contrapos-nn)
assume hindered $(\Gamma \ominus F (\text{ ?}\varepsilon', \text{ ?}h'))$
thus hindered $\text{ ?}\Omega$
proof(rule hindered-mono-web[rotated -1])
show weight $\text{ ?}\Omega z =$ weight $(\Gamma \ominus F (\text{ ?}\varepsilon', \text{ ?}h')) z$ **if** $z \notin A$ $(\Gamma \ominus F (\text{ ?}\varepsilon',$
 $\text{ ?}h'))$ **for** z
using *that* **unfolding** F'
apply(cases $z = y$)

```

    apply(simp-all add:  $\Omega$ -def minus-plus-current[ $OF\ fk\ g$ ]  $\Gamma'$ .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 ( $\Gamma \ominus F$  ( $? \varepsilon'$ ,  $?h'$ ))  $z \leq$  weight  $? \Omega\ z$  if  $z \in A$  ( $\Gamma \ominus F$ 
( $? \varepsilon'$ ,  $?h'$ )) for  $z$ 
    using that  $y$ - $B$  disjoint  $\delta$ -nonneg unfolding  $F'$ 
    apply(cases  $z = a$ )
    apply(simp-all add:  $\Omega$ -def minus-plus-current[ $OF\ fk\ g$ ]  $\Gamma'$ .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:  $\Omega$ -def)
    qed
    ultimately have ( $? \varepsilon'$ ,  $?h'$ )  $\in$  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  $\Gamma\ a$  by
simp
have sat  $\varepsilon h = \varepsilon h$  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  $\Gamma$  ( $F\ f$ ) and unhindered:  $\neg$  hindered ( $\Gamma \ominus F\ f$ )
by(cases  $f$ ; simp add: f unhindered'; fail)+
interpret  $\Gamma$ : 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  $?P$ - $k = \lambda k$ . 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 ( $\Gamma \ominus F\ f$ )  $k'$ ;  $k \leq k'$   $\rrbracket \implies k$ 
 $= k'$  by blast+

```

note $[simp] = \Gamma.weight-minus-web[OF\ k]$

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 $fk: current\ \Gamma\ ?fk$ **using** $f\ k$ **by**($rule\ current-plus-current-minus$)

have $d-OUT\ ?fk\ a \geq weight\ \Gamma\ a$
proof($rule\ ccontr$)
assume $\neg\ ?thesis$
hence $less: d-OUT\ ?fk\ a < weight\ \Gamma\ a$ **by** $simp$

define Ω **where** $\Omega = \Gamma \ominus F\ f \ominus k$
have $B-\Omega\ [simp]: B\ \Omega = B\ \Gamma$ **by**($simp\ add: \Omega-def$)

have $loose: loose\ \Omega$ **unfolding** $\Omega-def$ **using** $unhindered\ k\ k-w\ maximal$ **by**($rule\ \Gamma.loose-minus-web$)
interpret $\Omega: countable-bipartite-web\ \Omega$ **using** k **unfolding** $\Omega-def$
by($rule\ \Gamma.countable-bipartite-web-minus-web$)

have $a-\mathcal{E}: a \in TER_{\Gamma} \ominus F\ f\ k$ **using** $Field\ k\ k-w\ less$ **by**($rule\ a-TER$)
then have $weight\ \Omega\ a = weight\ \Gamma\ a - d-OUT\ (F\ f)\ a$
using $a\ disjoint$ **by**($auto\ simp\ add: roofed-circ-def\ \Omega-def\ SINK.simps$)
then have $weight-a: 0 < weight\ \Omega\ a$ **using** $less\ a-\mathcal{E}$
by($simp\ add: OUT-fk\ SINK.simps\ diff-gr0-ennreal$)

let $?P-y = \lambda y. y \in OUT_{\Omega}\ a \wedge weight\ \Omega\ y > 0$
define y **where** $y = Eps\ ?P-y$
have $Ex\ ?P-y$ **using** $Field\ k\ k-w\ less$ **unfolding** $\Omega-def$ **by**($rule\ Ex-y$)
hence $?P-y\ y$ **unfolding** $y-def$ **by**($rule\ someI-ex$)
hence $y \in OUT_{\Omega}\ a$ **and** $weight-y: weight\ \Omega\ y > 0$ **by** $blast+$
then have $y-B: y \in B\ \Omega$ **by**($auto\ simp\ add: outgoing-def\ \Omega-def\ dest: bipartite-E$)

define $bound$ **where** $bound = enn2real\ (\min\ (weight\ \Omega\ a)\ (weight\ \Omega\ y))$
have $bound-pos: bound > 0$ **using** $weight-y\ weight-a\ \Omega.weight-finite$
by($cases\ weight\ \Omega\ a\ weight\ \Omega\ y\ rule: ennreal2-cases$)
 $(simp-all\ add: bound-def\ min-def\ split: if-split-asm)$

let $?P-\delta = \lambda \delta. \delta > 0 \wedge \delta < bound \wedge \neg\ hindered\ (reduce-weight\ \Omega\ y\ \delta)$
define δ **where** $\delta = Eps\ ?P-\delta$
from $\Omega.unhinder[OF\ loose - weight-y\ bound-pos]\ y-B\ disjoint$ **have** $Ex\ ?P-\delta$
by($auto\ simp\ add: \Omega-def$)
hence $?P-\delta\ \delta$ **unfolding** $\delta-def$ **by**($rule\ someI-ex$)
hence $\delta-pos: 0 < \delta$ **by** $blast+$

let $?f' = (plus-current\ (fst\ f)\ (zero-current((a, y) := \delta)), plus-current\ (snd\ f)\ k)$


```

have sat:  $?f' = \text{sat } f$  using less by(simp add: sat-def k-def  $\Omega$ -def Let-def y-def
bound-def  $\delta$ -def split-def)
also have  $\dots = f$  unfolding f-def using fixp-above-unfold[OF zero] by simp
finally have  $\text{fst } ?f' (a, y) = \text{fst } f (a, y)$  by simp
hence  $\delta = 0$  using currentD-finite[OF  $\varepsilon$ -curr[OF Field']]  $\delta$ -pos
by(cases fst f (a, y)) simp-all
with  $\delta$ -pos show False by simp
qed

with currentD-weight-OUT[OF fk, of a] have  $d\text{-OUT } ?fk\ a = \text{weight } \Gamma\ a$  by
simp
moreover have current  $\Gamma\ ?fk$  using f k by(rule current-plus-current-minus)
moreover have  $\neg \text{hindered } (\Gamma \ominus ?fk)$  unfolding minus-plus-current[OF f k]
using unhindered k k-w by(rule  $\Gamma$ .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:  $\neg \text{hindered } \Gamma$ 
shows linkable  $\Gamma$ 
proof(cases A  $\Gamma = \{\}$ )
case True
thus ?thesis by(auto intro!: exI[where  $x = \text{zero-current}$ ] linkage.intros simp add:
web-flow-iff )
next
case nempty: False

```

```

let  $?P = \lambda f\ a\ f'. \text{current } (\Gamma \ominus f)\ f' \wedge d\text{-OUT } f'\ a = \text{weight } (\Gamma \ominus f)\ a \wedge \neg$ 
hindered  $(\Gamma \ominus f \ominus f')$ 

```

```

define enum where enum = from-nat-into (A  $\Gamma$ )
have enum-A: enum  $n \in A\ \Gamma$  for  $n$  using from-nat-into[OF nempty, of n]
by(simp add: enum-def)
have vertex-enum [simp]: vertex  $\Gamma\ (\text{enum } n)$  for  $n$  using enum-A[of n] A-vertex
by blast

```

```

define f where f = rec-nat zero-current ( $\lambda n\ f. \text{let } f' = \text{SOME } f'.\ ?P\ f\ (\text{enum } n)\ f' \text{ in plus-current } f\ f')$ 
have f-0 [simp]: f  $0 = \text{zero-current}$  by(simp add: f-def)
have f-Suc: f (Suc  $n$ ) = plus-current (f  $n$ ) (Eps ( $?P\ (f\ n)\ (\text{enum } n))$ ) for  $n$ 
by(simp add: f-def)

```

```

have f: current  $\Gamma\ (f\ n)$ 

```

```

and sat:  $\bigwedge m. m < n \implies d\text{-OUT } (f\ n) \ (enum\ m) = weight\ \Gamma \ (enum\ m)$ 
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  $\Gamma$ : 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  $\Gamma.unhindered\text{-saturate1}$ )(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)  $\leq weight\ \Gamma \ (enum\ m)$  using f'' by(rule currentD-weight-OUT)
        moreover have weight  $\Gamma \ (enum\ m) \leq d\text{-OUT } (f\ (Suc\ n)) \ (enum\ m)$ 
        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)
          case False
            hence m < n using 2 by simp
            thus ?thesis using Suc.IH(2)[OF  $\langle m < n \rangle$ ] 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  $\Gamma$ : 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  $\Gamma.unhindered\text{-saturate1}$ )(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 ( $\leq$ ) (range f) by(rule incseq-chain-range)

define g where g = Sup (range f)
have support-flow g  $\subseteq$  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  $\Gamma$  g unfolding g-def by(rule current-Sup)(auto
simp add: f)
moreover
have d-OUT g x = weight  $\Gamma$  x if x  $\in$  A  $\Gamma$  for x
proof(rule antisym)
show d-OUT g x  $\leq$  weight  $\Gamma$  x using g by(rule currentD-weight-OUT)
have countable (A  $\Gamma$ ) 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  $\geq$  weight  $\Gamma$  x by simp
then show weight  $\Gamma$  x  $\leq$  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  $\Gamma$ 
shows linkable  $\Gamma$ 
proof —
interpret bi: countable-bipartite-web bipartite-web-of  $\Gamma$  by(rule countable-bipartite-web-of)
have  $\neg$  hindered (bipartite-web-of  $\Gamma$ ) using assms by(rule unhindered-bipartite-web-of)
then have linkable (bipartite-web-of  $\Gamma$ )
by(rule bi.unhindered-linkable)
then show ?thesis by(rule linkable-bipartite-web-ofD) simp
qed

lemma ex-orthogonal-current: — Lemma 4.15
 $\exists$  f S. web-flow  $\Gamma$  f  $\wedge$  separating  $\Gamma$  S  $\wedge$  orthogonal-current  $\Gamma$  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*: $\bigwedge x. \neg \text{edge } \Delta x \text{ (source } \Delta)$
 and *sink-out*: $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$
 and *undead*: $\bigwedge x y. \text{edge } \Delta x y \implies (\exists z. \text{edge } \Delta y z) \vee (\exists z. \text{edge } \Delta z x)$
 and *source-sink*: $\neg \text{edge } \Delta (\text{source } \Delta) (\text{sink } \Delta)$
 and *no-loop*: $\bigwedge x. \neg \text{edge } \Delta x x$
 and *capacity-pos*: $\bigwedge e. e \in \mathbf{E} \implies \text{capacity } \Delta e > 0$
 shows $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$
 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*: $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$
 and *source-in*: $\bigwedge x. \neg \text{edge } \Delta x \text{ (source } \Delta)$
 and *no-loop*: $\bigwedge x. \neg \text{edge } \Delta x x$
 and *capacity-pos*: $\bigwedge e. e \in \mathbf{E} \implies \text{capacity } \Delta e > 0$
 shows $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$
 proof –
 interpret *antiparallel-edges* $\Delta ..$
 interpret Δ'' : *countable-network* Δ'' by(*rule* Δ'' -*countable-network*)
 have $\exists f S. \text{flow } \Delta'' f \wedge \text{cut } \Delta'' S \wedge \text{orthogonal } \Delta'' f S$
 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* $\Delta'' f$ and *cut*: *cut* $\Delta'' S$ and *ortho*: *orthogonal* $\Delta'' f S$ 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*: $\bigwedge y. \neg \text{edge } \Delta (\text{sink } \Delta) y$
 and *source-in*: $\bigwedge x. \neg \text{edge } \Delta x \text{ (source } \Delta)$
 and *capacity-pos*: $\bigwedge e. e \in \mathbf{E} \implies \text{capacity } \Delta e > 0$
 shows $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$
 proof –
 interpret *antiparallel-edges* $\Delta ..$
 interpret Δ'' : *countable-network* Δ'' by(*rule* Δ'' -*countable-network*)
 have $\exists f S. \text{flow } \Delta'' f \wedge \text{cut } \Delta'' S \wedge \text{orthogonal } \Delta'' f S$
 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* $\Delta'' f$ and *cut*: *cut* $\Delta'' S$ and *ortho*: *orthogonal* $\Delta'' f S$ by *blast*
 have *flow* Δ (*collect* f) using f by(*rule* *flow-collect*)

moreover have $\text{cut } \Delta \text{ (cut' } S)$ using cut by (rule cut-cut')
 moreover have $\text{orthogonal } \Delta \text{ (collect } f) \text{ (cut' } S)$ using $\text{ortho } f$ by (rule orthogonal-cut')
 ultimately show $?thesis$ by blast
 qed

theorem *max-flow-min-cut*:
 $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$
proof –
 interpret Δ'' : *countable-network* Δ'' by (rule Δ'' -countable-network)
 have $\exists f S. \text{flow } \Delta'' f \wedge \text{cut } \Delta'' S \wedge \text{orthogonal } \Delta'' f S$ by (rule Δ'' .max-flow-min-cut'')
 auto
 then obtain $f S$ where f : $\text{flow } \Delta'' f$ and cut : $\text{cut } \Delta'' S$ and ortho : $\text{orthogonal } \Delta'' f S$ by blast
 from $\text{flow-}\Delta''$ [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]: $\text{countable } \mathbf{E}$
 and *source-neq-sink* [simp]: $\text{source } \Delta \neq \text{sink } \Delta$
 and *capacity-outside*: $\forall e. e \notin \mathbf{E} \longrightarrow \text{capacity } \Delta e = 0$
 and *capacity-finite* [simp]: $\forall e. \text{capacity } \Delta e \neq \top$
 shows $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$
proof –
 interpret *countable-network* Δ using *assms* by (unfold-locale) auto
 show $?thesis$ by (rule max-flow-min-cut)
 qed

hide-const (open) $A B \text{ weight}$

 end

theory *Rel-PMF-Characterisation* imports

Matrix-For-Marginals
begin

12 Characterisation of *rel-pmf*

proposition *rel-pmf-measureI*:

fixes $p :: 'a \text{ pmf}$ **and** $q :: 'b \text{ pmf}$
assumes $le: \bigwedge A. \text{measure} (\text{measure-pmf } p) A \leq \text{measure} (\text{measure-pmf } q) \{y. \exists x \in A. R \ x \ y\}$
shows *rel-pmf* $R \ p \ q$
proof –
let $?A = \text{set-pmf } p$ **and** $?f = \lambda x. \text{ennreal} (\text{pmf } p \ x)$
and $?B = \text{set-pmf } q$ **and** $?g = \lambda y. \text{ennreal} (\text{pmf } q \ y)$
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 $(\sum^+ x \in X. ?f \ x) \leq (\sum^+ y \in R' \text{ `` } X. ?g \ y)$ (**is** $?lhs \leq ?rhs$) **if** $X \subseteq \text{set-pmf } p$
for X
proof –
have $?lhs = \text{measure} (\text{measure-pmf } p) \ X$
by(*simp add: nn-integral-pmf measure-pmf.emmeasure-eq-measure*)
also have $\dots \leq \text{measure} (\text{measure-pmf } q) \{y. \exists x \in X. R \ x \ y\}$ **by**(*simp add: le*)
also have $\dots = \text{measure} (\text{measure-pmf } q) (R' \text{ `` } X)$ **using** *that*
by(*auto 4 3 simp add: R'-def AE-measure-pmf-iff intro! : measure-eq-AE*)
also have $\dots = ?rhs$ **by**(*simp add: nn-integral-pmf measure-pmf.emmeasure-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*: $\bigwedge x \ y. 0 < h \ x \ y \implies (x, y) \in R'$
and *bound*: $\bigwedge x \ y. h \ x \ y \neq \top$
and $p: \bigwedge x. x \in ?A \implies (\sum^+ y \in ?B. h \ x \ y) = ?f \ x$
and $q: \bigwedge y. y \in ?B \implies (\sum^+ x \in ?A. h \ x \ y) = ?g \ y$
by(*rule bounded-matrix-for-marginals-ennreal*) *blast*+

let $?z = \lambda(x, y). \text{enn2real} (h \ x \ y)$
define z **where** $z = \text{embed-pmf } ?z$
have *nonneg*: $\bigwedge xy. 0 \leq ?z \ xy$ **by** *clarsimp*
have *outside*: $h \ x \ y = 0$ **if** $x \notin \text{set-pmf } p \vee y \notin \text{set-pmf } q \vee \neg R \ x \ y$ **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 ... = ( $\sum^+ x. (\sum^+ y. h x y)$ ) using bound
    by (simp add: nn-integral-count-space-reindex ennreal-enn2real-if)
  also have ... = ( $\sum^+ x \in ?A. (\sum^+ y \in ?B. h x y)$ )
    by (auto intro!: nn-integral-cong nn-integral-zero' simp add: nn-integral-count-space-indicator
outside split: split-indicator)
  also have ... = ( $\sum^+ x \in ?A. ?f x$ ) by (auto simp add: p intro!: nn-integral-cong)
  also have ... = ( $\sum^+ x. ?f x$ )
    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 = ( $\sum^+ e \in \text{range } (Pair x). \text{pmf } z e$ )
      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 ... = ( $\sum^+ y. h x y$ )
      using bound by (simp add: nn-integral-count-space-reindex ennreal-enn2real-if)
    also have ... = ( $\sum^+ y \in ?B. h x y$ ) using outside
      by (auto simp add: nn-integral-count-space-indicator intro!: nn-integral-cong
split: split-indicator)
    also have ... = ?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 = ( $\sum^+ e \in \text{range } (\lambda x. (x, y)). \text{pmf } z e$ )
      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 ... = ( $\sum^+ x. h x y$ )
      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 \in 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:  $\bigwedge A. A \subseteq \text{set-pmf } p \implies \text{measure-pmf.prob } p\ A \leq \text{measure-pmf.prob}$ 
 $q\ \{y \in \text{set-pmf } q. \exists x \in A. R\ x\ y\}$ 
  shows rel-pmf R p q
proof(rule rel-pmf-measureI)
  fix A
  let ?A =  $A \cap \text{set-pmf } p$ 
  have  $\text{measure-pmf.prob } p\ A = \text{measure-pmf.prob } p\ ?A$  by(simp add: measure-Int-set-pmf)
  also have ...  $\leq \text{measure-pmf.prob } q\ \{y \in \text{set-pmf } q. \exists x \in ?A. R\ x\ y\}$  by(rule le)
simp
  also have ...  $\leq \text{measure-pmf.prob } q\ \{y. \exists x \in A. R\ x\ y\}$ 
  by(rule measure-pmf.finite-measure-mono) auto
  finally show  $\text{measure-pmf.prob } p\ A \leq \dots$  .
qed

```

lemma *rel-pmf-code* [*code*]:

```

   $\text{rel-pmf } R\ p\ q \longleftrightarrow$ 
  (let B = set-pmf q in
 $\forall A \in \text{Pow } (\text{set-pmf } p). \text{measure-pmf.prob } p\ A \leq \text{measure-pmf.prob } q\ (\text{snd } ' \text{Set.filter } (\text{case-prod } R)\ (A \times B)))$ )
  unfolding Let-def
proof(intro iffI strip)
  have eq:  $\text{snd } ' \text{Set.filter } (\text{case-prod } R)\ (A \times \text{set-pmf } q) = \{y. \exists x \in A. R\ x\ y\} \cap \text{set-pmf } q$  for A
  by(auto intro: rev-image-eqI simp add: Set.filter-def)
  show  $\text{measure-pmf.prob } p\ A \leq \text{measure-pmf.prob } q\ (\text{snd } ' \text{Set.filter } (\text{case-prod } R)\ (A \times \text{set-pmf } q))$ 
  if rel-pmf R p q and  $A \in \text{Pow } (\text{set-pmf } p)$  for A
  using that by(auto dest: rel-pmf-measureD simp add: eq measure-Int-set-pmf)
  show rel-pmf R p q if  $\forall A \in \text{Pow } (\text{set-pmf } p). \text{measure-pmf.prob } p\ A \leq \text{measure-pmf.prob } q\ (\text{snd } ' \text{Set.filter } (\text{case-prod } R)\ (A \times \text{set-pmf } q))$ 
  using that by(intro rel-pmf-measureI')(auto intro: ord-le-eq-trans arg-cong2[where f=measure] simp add: eq)
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  $\Rightarrow$  'b  $\Rightarrow$  bool begin

```

```

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

```

```

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

```

```

private inductive-cases edge'-SourceE [elim!]: edge' Source yv

```

```

private inductive-cases edge'-LeftE [elim!]: edge' (Left x) yv

```

```

private inductive-cases edge'-RightE [elim!]: edge' xv (Right y)

```

```

private inductive-cases edge'-SinkE [elim!]: edge' xv Sink

```

```

private function cap :: ('a, 'b) vertex flow where

```

```

  cap (xv, Left x) = (if xv = Source then ennreal (pmf p x) else 0)

```

```

| cap (Left x, Right y) =

```

```

  (if R x y  $\wedge$  x  $\in$  set-pmf p  $\wedge$  y  $\in$  set-pmf q

```

```

    then pmf q y — Return pmf q y so that total weight of x's neighbours is finite,
    i.e., the network satisfies bounded-countable-network.

```

```

    else 0)
| cap (Right y, yv) = (if yv = Sink then ennreal (pmf q y) else 0)
| cap (Source, Right y) = 0
| cap (Source, Sink) = 0
| cap (xv, Source) = 0
| cap (Sink, yv) = 0
| cap (Left x, Sink) = 0
  by pat-completeness auto
termination by lexicographic-order

```

private definition $\Delta :: ('a, 'b)$ vertex network
 where $\Delta = (\text{edge} = \text{edge}', \text{capacity} = \text{cap}, \text{source} = \text{Source}, \text{sink} = \text{Sink})$

private lemma Δ -sel [simp]:
 edge $\Delta = \text{edge}'$
 capacity $\Delta = \text{cap}$
 source $\Delta = \text{Source}$
 sink $\Delta = \text{Sink}$
 by (simp-all add: Δ -def)

private lemma IN-Left [simp]: $\text{IN}_\Delta (\text{Left } x) = (\text{if } x \in \text{set-pmf } p \text{ then } \{\text{Source}\} \text{ else } \{\})$
 by (auto simp add: incoming-def)
private lemma OUT-Right [simp]: $\text{OUT}_\Delta (\text{Right } y) = (\text{if } y \in \text{set-pmf } q \text{ then } \{\text{Sink}\} \text{ else } \{\})$
 by (auto simp add: outgoing-def)

interpretation network: countable-network Δ

proof

```

  show source  $\Delta \neq$  sink  $\Delta$  by simp
  show capacity  $\Delta \ e = 0$  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  $\Delta \ e \neq$  top for  $e$  by (cases  $e$  rule: cap.cases)(auto)
  have  $\mathbf{E}_\Delta \subseteq ((\text{Pair Source} \circ \text{Left}) \text{ ` set-pmf } p) \cup (\text{map-prod Left Right ` (set-pmf } p \times \text{set-pmf } q)) \cup ((\lambda y. (\text{Right } y, \text{Sink})) \text{ ` set-pmf } q)$ 
    by (auto elim: edge'.cases)
  thus countable  $\mathbf{E}_\Delta$  by (rule countable-subset) auto
qed

```

private lemma OUT-cap-Source: $d\text{-OUT cap Source} = 1$

proof –

```

  have  $d\text{-OUT cap Source} = (\sum^+ y \in \text{range Left. cap (Source, y))$ 
    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  $\dots = (\sum^+ y. \text{pmf } p \ y)$  by (simp add: nn-integral-count-space-reindex)
  also have  $\dots = 1$  by (simp add: nn-integral-pmf)
  finally show ?thesis .

```

qed

private lemma IN-cap-Left: $d\text{-IN cap (Left } x) = \text{pmf } p \ x$

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: $\exists f S. \text{flow } \Delta f \wedge \text{cut } \Delta S \wedge \text{orthogonal } \Delta f S$
and le: $\bigwedge A. \text{measure } (\text{measure-pmf } p) A \leq \text{measure } (\text{measure-pmf } q) \{y. \exists x \in A. R x y\}$
shows rel-pmf R p q
proof –
from ex-flow **obtain** f S
where f: flow Δf **and** cut: cut ΔS **and** ortho: orthogonal $\Delta f S$ **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 $\Delta f \leq 1$ **using** flowD-capacity-OUT[OF f, of Source] **by**(simp add: OUT-cap-Source)
moreover **have** $1 \leq \text{value-flow } \Delta f$
proof –
let ?L = Left - ' S $\cap$ set-pmf p
let ?R' = $\{y \mid y x. x \in \text{set-pmf } p \wedge \text{Left } x \in S \wedge R x y \wedge y \in \text{set-pmf } q \wedge \text{Right } y \in S\}$
let ?R'' = $\{y \mid y x. x \in \text{set-pmf } p \wedge \text{Left } x \in S \wedge R x y \wedge y \in \text{set-pmf } q \wedge \neg \text{Right } y \in S\}$
have value-flow $\Delta f = (\sum^+ x \in \text{range Left}. f (\text{Source}, x))$ **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** $\dots = (\sum^+ x. f (\text{Source}, \text{Left } x) * \text{indicator } ?L x) + (\sum^+ x. f (\text{Source}, \text{Left } x) * \text{indicator } (- ?L) x)$
by(subst nn-integral-add[symmetric])(auto simp add: nn-integral-count-space-reindex intro!: nn-integral-cong split: split-indicator)
also **have** $(\sum^+ x. f (\text{Source}, \text{Left } x) * \text{indicator } (- ?L) x) = (\sum^+ x \in - ?L. \text{cap } (\text{Source}, \text{Left } x))$
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 network.flowD-outside[OF f])

```

done
also have ... = ( $\sum^+ x \in ?L. \text{pmf } p \ x$ ) by simp
also have ... =  $\text{emeasure } p \ (- \ ?L)$  by (simp add: nn-integral-pmf)
also have ( $\sum^+ x. f \ (\text{Source}, \text{Left } x) * \text{indicator } ?L \ x$ ) = ( $\sum^+ x \in ?L. d\text{-IN } f \ (\text{Left } x)$ )
by (subst d-IN-alt-def[of -  $\Delta$ ]) (auto simp add: network.flowD-outside[OF f]
nn-integral-count-space-indicator intro!: nn-integral-cong)
also have ... = ( $\sum^+ x \in ?L. d\text{-OUT } f \ (\text{Left } x)$ )
by (rule nn-integral-cong flowD-KIR[OF f, symmetric]) + simp-all
also have ... = ( $\sum^+ x. \sum^+ y. f \ (\text{Left } x, y) * \text{indicator } (\text{range } \text{Right}) \ y * \text{indicator } ?L \ x$ )
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 ... = ( $\sum^+ y \in \text{range } \text{Right}. \sum^+ x. f \ (\text{Left } x, y) * \text{indicator } ?L \ x$ )
by (subst nn-integral-fst-count-space[where  $f = \text{case-prod } -, \text{ simplified}$ ])
(simp add: nn-integral-snd-count-space[where  $f = \text{case-prod } -, \text{ simplified}$ ])
nn-integral-count-space-indicator nn-integral-cmult[symmetric] mult-ac)
also have ... = ( $\sum^+ y. \sum^+ x. f \ (\text{Left } x, \text{Right } y) * \text{indicator } ?L \ x$ )
by (simp add: nn-integral-count-space-reindex)
also have ... = ( $\sum^+ y. \sum^+ x. f \ (\text{Left } x, \text{Right } y) * \text{indicator } ?L \ x * \text{indicator } ?R' \ y$ )
+
( $\sum^+ y. \sum^+ x. f \ (\text{Left } x, \text{Right } y) * \text{indicator } ?L \ x * \text{indicator } ?R'' \ y$ )
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 ( $\sum^+ y. \sum^+ x. f \ (\text{Left } x, \text{Right } y) * \text{indicator } ?L \ x * \text{indicator } ?R' \ y$ ) =
( $\sum^+ y. \sum^+ x. f \ (\text{Left } x, \text{Right } y) * \text{indicator } ?R' \ y$ )
apply (clarsimp simp add: network.flowD-outside[OF f] intro!: nn-integral-cong
split: split-indicator)
subgoal for  $y \ x$  by (cases edge  $\Delta \ (\text{Left } x) \ (\text{Right } y)$ ) (auto intro: orthogonalD-in[OF ortho] network.flowD-outside[OF f])
done
also have ... = ( $\sum^+ y. \sum^+ x \in \text{range } \text{Left}. f \ (x, \text{Right } y) * \text{indicator } ?R' \ y$ )
by (simp add: nn-integral-count-space-reindex)
also have ... = ( $\sum^+ y \in ?R'. d\text{-IN } f \ (\text{Right } y)$ )
by (subst d-IN-alt-def[of -  $\Delta$ ]) (auto simp add: network.flowD-outside[OF f]
nn-integral-count-space-indicator incoming-def intro!: nn-integral-cong split: split-indicator)
also have ... = ( $\sum^+ y \in ?R'. d\text{-OUT } f \ (\text{Right } y)$ ) using flowD-KIR[OF f] by
simp
also have ... = ( $\sum^+ y \in ?R'. d\text{-OUT } \text{cap } (\text{Right } y)$ )
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 ... = ( $\sum^+ y \in ?R'. \text{pmf } q \ y$ ) by (simp add: OUT-cap-Right)
also have ... =  $\text{emeasure } q \ ?R'$  by (simp add: nn-integral-pmf)
also have ( $\sum^+ y. \sum^+ x. f \ (\text{Left } x, \text{Right } y) * \text{indicator } ?L \ x * \text{indicator } ?R'' \ y$ )  $\geq \text{emeasure } q \ ?R''$  (is  $?lhs \geq -$ )
proof -
have  $?lhs = (\sum^+ y. \sum^+ x. (\text{if } R \ x \ y \text{ then } \text{pmf } q \ y \text{ else } 0) * \text{indicator } ?L \ x$ 

```

```

* 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 ...  $\geq (\sum^+ y. \text{pmf } q \ y * \text{indicator } ?R'' \ y)$ 
  by(rule nn-integral-mono)(auto split: split-indicator intro: order-trans[OF -
nn-integral-ge-point])
  also have  $(\sum^+ y. \text{pmf } q \ y * \text{indicator } ?R'' \ y) = (\sum^+ y \in ?R''. \text{pmf } q \ y)$ 
  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  $\Delta \ f \geq \text{emeasure } q \ ?R' + \text{emeasure } q \ ?R'' +$ 
emeasure  $p \ (- \ ?L)$ 
  by(simp add: add-right-mono)
  also have  $\text{emeasure } q \ ?R' + \text{emeasure } q \ ?R'' = \text{emeasure } q \ \{y \mid y \ x. \ x \in \text{set-pmf}$ 
 $p \wedge \text{Left } x \in S \wedge R \ x \ y \wedge y \in \text{set-pmf } q\}$ 
  by(subst plus-emeasure)(auto intro!: arg-cong2[where f=emeasure])
  also have ...  $\geq \text{emeasure } p \ ?L$  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  $\Delta \ f \geq \text{emeasure } (\text{measure-pmf } p) \ ?L + \text{emeasure}$ 
 $(\text{measure-pmf } p) \ (- \ ?L)$ 
  by(smt (verit, best) add-right-mono inf.absorb-iff2 le-inf-iff)
  also have  $\text{emeasure } (\text{measure-pmf } p) \ ?L + \text{emeasure } (\text{measure-pmf } p) \ (- \ ?L)$ 
 $= \text{emeasure } (\text{measure-pmf } p) \ (?L \cup - \ ?L)$ 
  by(subst plus-emeasure) auto
  also have  $?L \cup - \ ?L = \text{UNIV}$  by blast
  finally show ?thesis by simp
qed
ultimately have val: value-flow  $\Delta \ f = 1$  by simp

have SAT-p:  $f \ (\text{Source}, \text{Left } x) = \text{pmf } p \ x$  for  $x$ 
proof(rule antisym)
  show  $f \ (\text{Source}, \text{Left } x) \leq \text{pmf } p \ x$  using flowD-capacity[OF f, of (Source, Left
x)] by simp
  show  $\text{pmf } p \ x \leq f \ (\text{Source}, \text{Left } x)$ 
  proof(rule ccontr)
    assume *:  $\neg ?thesis$ 
    have finite:  $(\sum^+ y. f \ (\text{Source}, \text{Left } y) * \text{indicator } (- \ \{x\}) \ y) \neq \infty$ 
    proof -
      have  $(\sum^+ y. f \ (\text{Source}, \text{Left } y) * \text{indicator } (- \ \{x\}) \ y) \leq (\sum^+ y \in \text{range}$ 
 $\text{Left}. f \ (\text{Source}, y))$ 
      by(auto simp add: nn-integral-count-space-reindex intro!: nn-integral-mono
split: split-indicator)
      also have ...  $= \text{value-flow } \Delta \ f$ 
      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
  qed

```

have *value-flow* $\Delta f = (\sum^+ y \in \text{range Left. } f \text{ (Source, } y))$
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 $\dots = (\sum^+ y. f \text{ (Source, Left } y) * \text{indicator } (- \{x\}) y) + (\sum^+$
 $y. f \text{ (Source, Left } y) * \text{indicator } \{x\} y)$
by(*subst nn-integral-add[symmetric]*)(*auto simp add: nn-integral-count-space-reindex*
intro! nn-integral-cong split: split-indicator)
also have $\dots < (\sum^+ y. f \text{ (Source, Left } y) * \text{indicator } (- \{x\}) y) + (\sum^+$
 $y. \text{pmf } p y * \text{indicator } \{x\} y)$ **using** ** finite*
by(*auto simp add:*)
also have $\dots \leq (\sum^+ y. \text{pmf } p y * \text{indicator } (- \{x\}) y) + (\sum^+ y. \text{pmf } p y$
 $* \text{indicator } \{x\} y)$
using *flowD-capacity[OF f, of (Source, Left -)]*
by(*auto intro! nn-integral-mono split: split-indicator*)
also have $\dots = (\sum^+ y. \text{pmf } p y)$
by(*subst nn-integral-add[symmetric]*)(*auto intro! nn-integral-cong split:*
split-indicator)
also have $\dots = 1$ **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* $= (\sum^+ x \in \text{range Right. } f \text{ (x, Sink)})$ **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 $\dots = (\sum^+ y. d\text{-OUT } f \text{ (Right } y))$ **by**(*simp add: nn-integral-count-space-reindex*
OUT-f-Right)
also have $\dots = (\sum^+ y. d\text{-IN } f \text{ (Right } y))$ **by**(*simp add: flowD-KIR[OF f]*)
also have $\dots = (\sum^+ y. (\sum^+ x \in \text{range Left. } f \text{ (x, Right } y)))$
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 $\dots = (\sum^+ y. \sum^+ x. f \text{ (Left } x, \text{Right } y))$ **by**(*simp add: nn-integral-count-space-reindex*)
also have $\dots = (\sum^+ x. \sum^+ y. f \text{ (Left } x, \text{Right } y))$
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 $\dots = (\sum^+ x. (\sum^+ y \in \text{range Right. } f \text{ (Left } x, y)))$
by(*simp add: nn-integral-count-space-reindex*)
also have $\dots = (\sum^+ x. d\text{-OUT } f \text{ (Left } x))$ **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 $\dots = (\sum^+ x. d\text{-IN } f \text{ (Left } x))$ **by**(*simp add: flowD-KIR[OF f]*)
also have $\dots = (\sum^+ x. \text{pmf } p x)$ **by**(*simp add: IN-f-Left SAT-p*)
also have $\dots = 1$ **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  $f \text{ (Right } y, \text{ Sink)} \leq \text{pmf } q \text{ } y$  using flowD-capacity[OF  $f$ , of (Right  $y$ , Sink)] by simp
  show  $\text{pmf } q \text{ } y \leq f \text{ (Right } y, \text{ Sink)}$ 
  proof(rule ccontr)
    assume *:  $\neg ?thesis$ 
    have finite:  $(\sum^+ x. f \text{ (Right } x, \text{ Sink)} * \text{indicator } (- \{y\}) x) \neq \infty$ 
    proof -
      have  $(\sum^+ x. f \text{ (Right } x, \text{ Sink)} * \text{indicator } (- \{y\}) x) \leq (\sum^+ x \in \text{range Right. } f \text{ (} x, \text{ Sink)})$ 
      by(auto simp add: nn-integral-count-space-reindex intro!: nn-integral-mono split: split-indicator)
    also have  $\dots = d\text{-IN } f \text{ Sink}$ 
    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  $d\text{-IN } f \text{ Sink} = (\sum^+ x \in \text{range Right. } f \text{ (} x, \text{ Sink)})$ 
  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  $\dots = (\sum^+ x. f \text{ (Right } x, \text{ Sink)} * \text{indicator } (- \{y\}) x) + (\sum^+ x. f \text{ (Right } x, \text{ Sink)} * \text{indicator } \{y\} x)$ 
  by(subst nn-integral-add[symmetric])(auto simp add: nn-integral-count-space-reindex intro!: nn-integral-cong split: split-indicator)
  also have  $\dots < (\sum^+ x. f \text{ (Right } x, \text{ Sink)} * \text{indicator } (- \{y\}) x) + (\sum^+ x. \text{pmf } q \text{ } x * \text{indicator } \{y\} x)$  using * finite
  by auto
  also have  $\dots \leq (\sum^+ x. \text{pmf } q \text{ } x * \text{indicator } (- \{y\}) x) + (\sum^+ x. \text{pmf } q \text{ } x * \text{indicator } \{y\} x)$ 
  using flowD-capacity[OF  $f$ , of (Right -, Sink)]
  by(auto intro!: nn-integral-mono split: split-indicator)
  also have  $\dots = (\sum^+ x. \text{pmf } q \text{ } x)$ 
  by(subst nn-integral-add[symmetric])(auto intro!: nn-integral-cong split: split-indicator)
  also have  $\dots = 1$  unfolding nn-integral-pmf by simp
  finally show False using IN-Sink by simp
qed
qed

let ?z =  $\lambda(x, y). \text{enn2real } (f \text{ (Left } x, \text{ Right } y))$ 
have nonneg:  $\bigwedge xy. 0 \leq ?z \text{ } xy$  by clarsimp
have prob:  $(\sum^+ xy. ?z \text{ } xy) = 1$ 
proof -
  have  $(\sum^+ xy. ?z \text{ } xy) = (\sum^+ x. \sum^+ y. (\text{ennreal } \circ ?z) (x, y))$ 
  unfolding nn-integral-fst-count-space by(simp add: split-def o-def)
  also have  $\dots = (\sum^+ x. (\sum^+ y \in \text{range Right. } f \text{ (Left } x, y)))$ 
  by(auto simp add: nn-integral-count-space-reindex intro!: nn-integral-cong)
  also have  $\dots = (\sum^+ x. d\text{-OUT } f \text{ (Left } x))$ 
  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 ... = ( $\sum^+ x. d\text{-IN } f \text{ (Left } x)$ ) using flowD-KIR[OF f] by simp
  also have ... = ( $\sum^+ x \in \text{range Left. } f \text{ (Source, } x)$ ) by (simp add: nn-integral-count-space-reindex
IN-f-Left)
  also have ... = value-flow  $\Delta f$ 
  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)  $\in$  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 = ( $\sum^+ e \in \text{range (Pair } x). \text{ pmf } z \text{ } e$ )
    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 ... = ( $\sum^+ y \in \text{range Right. } f \text{ (Left } x, y)$ ) 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 = ( $\sum^+ e \in \text{range } (\lambda x. (x, y)). \text{ pmf } z \text{ } e$ )
    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 ... = ( $\sum^+ x \in \text{range Left. } f \text{ (} x, \text{ Right } y)$ ) 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)
  qed

```


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: $\bigwedge A. \text{measure } (\text{measure-pmf } p) A \leq \text{measure } (\text{measure-pmf } q) \{y. \exists x \in A. R x y\}$
 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) ≤ 1 **for** x
 proof –
 have *d-OUT cap* (Left x) $\leq (\sum^+ y \in \text{range } \text{Right}. \text{cap } (\text{Left } x, y))$
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 ... = $(\sum^+ y. \text{cap } (\text{Left } x, \text{Right } y))$ **by**(simp add: *nn-integral-count-space-reindex*)
 also have ... $\leq (\sum^+ y. \text{pmf } q y)$ **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** $x \in \mathbf{V}_\Delta$ $x \neq \text{source } \Delta$ $x \neq \text{sink } \Delta$ **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: $\bigwedge A. \text{measure } (\text{measure-pmf } p) A \leq \text{measure } (\text{measure-pmf } q) \{y. \exists x \in A. R x y\}$
 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  $pq$ :  $\text{rel-pmf } R \ p \ q$  and  $qr$ :  $\text{rel-pmf } S \ q \ r$ 
have  $\text{measure } (\text{measure-pmf } p) \ A \leq \text{measure } (\text{measure-pmf } q) \ \{y. \exists x \in A. \ R \ x \ y\}$ 
  (is  $- \leq \text{measure} - ?B$ ) using  $pq$  by ( $\text{rule rel-pmf-measureD}$ )
also have  $\dots \leq \text{measure } (\text{measure-pmf } r) \ \{z. \exists y \in ?B. \ S \ y \ z\}$ 
  using  $qr$  by ( $\text{rule rel-pmf-measureD}$ )
also have  $\{z. \exists y \in ?B. \ S \ y \ z\} = \{z. \exists x \in A. \ (R \ O O \ S) \ x \ z\}$  by auto
finally show  $\text{measure } (\text{measure-pmf } p) \ A \leq \text{measure } (\text{measure-pmf } r) \ \dots$  .
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.