blockDAGs

Jörn

August 13, 2021

Contents

theory *DAGs*
imports *Main Graph-Theory.Graph-Theory*
begin

1 DAG

locale *DAG* = *digraph* +
assumes *cycle-free*: $\neg(v \to^{+}{}_{G}\ v)$

sublocale *DAG* ⊆ *wf-digraph* **using** *DAG-def digraph-def nomulti-digraph-def DAG-axioms* **by** *auto*

1.1 Functions and Definitions

fun *direct-past*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a \Rightarrow$ $'a$ *set*
where *direct-past G a* = $\{b \in$ *verts G*. $(a,b) \in$ *arcs-ends G*$\}$

fun *future-nodes*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a \Rightarrow$ $'a$ *set*
where *future-nodes G a* = $\{b \in$ *verts G*. $b \to^{+}{}_{G}\ a\}$

fun *past-nodes*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a \Rightarrow$ $'a$ *set*
where *past-nodes G a* = $\{b \in$ *verts G*. $a \to^{+}{}_{G}\ b\}$

fun *past-nodes-refl* :: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a \Rightarrow$ $'a$ *set*
where *past-nodes-refl G a* = $\{b \in$ *verts G*. $a \to^{*}{}_{G}\ b\}$

fun *anticone*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a \Rightarrow$ $'a$ *set*
where *anticone G a* = $\{b \in$ *verts G*. $\neg(a \to^{+}{}_{G}\ b \lor b \to^{+}{}_{G}\ a \lor a = b)\}$

fun *reduce-past*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a \Rightarrow$ $('a,'b)$ *pre-digraph*
where
reduce-past G a = *induce-subgraph G* (*past-nodes G a*)

fun *reduce-past-refl*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a \Rightarrow$ $('a,'b)$ *pre-digraph*
where
reduce-past-refl G a = *induce-subgraph G* (*past-nodes-refl G a*)

fun *is-tip*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a \Rightarrow$ *bool*
where *is-tip G a* = $((a \in$ *verts G*$) \land (\forall\ x \in$ *verts G*. $\neg\ x \to^{+}{}_{G}\ a))$

definition *tips*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ $'a$ *set*
where *tips G* = $\{v \in$ *verts G*. *is-tip G v*$\}$

fun *kCluster*:: $('a,'b)$ *pre-digraph* $\Rightarrow$ *nat* $\Rightarrow$ $'a$ *set* $\Rightarrow$ *bool*
where *kCluster G k C* = (*if* ($C \subseteq$ (*verts G*))
then ($\forall\, a \in C$. *card* ((*anticone G a*) $\cap\ C) \leq k$) *else False*)

1.2 Lemmas

lemma (**in** *DAG*) *unidirectional*:
$u \rightarrow^{+}{}_{G}\ v \longrightarrow \neg(\ v \rightarrow^{*}{}_{G}\ u)$
using *cycle-free reachable1-reachable-trans* **by** *auto*

1.2.1 Tips

lemma (**in** *DAG*) *tips-not-referenced*:
assumes *is-tip G t*
shows $\forall\, x.\ \neg\ x \rightarrow^{+}\ t$
using *is-tip.simps assms reachable1-in-verts*(*1*)
by *metis*

lemma (**in** *DAG*) *del-tips-dag*:
assumes *is-tip G t*
shows *DAG* (*del-vert t*)
unfolding *DAG-def DAG-axioms-def*
proof *safe*
show *digraph* (*del-vert t*) **using** *del-vert-simps DAG-axioms*
digraph-def
using *digraph-subgraph subgraph-del-vert*
by *auto*
next
fix *v*
assume $v \rightarrow^{+}{}_{\mathit{del\text{-}vert}\ t}\ v$
then have $v \rightarrow^{+}\ v$ **using** *subgraph-del-vert*
by (*meson arcs-ends-mono trancl-mono*)
then show *False*
by (*simp add*: *cycle-free*)
qed

lemma (**in** *digraph*) *tips-finite*:
shows *finite* (*tips G*)
using *tips-def fin-digraph.finite-verts digraph.axioms*(*1*) *digraph-axioms Collect-mono is-tip.simps*
by (*simp add*: *tips-def*)

lemma (**in** *digraph*) *tips-in-verts*:
shows *tips G* $\subseteq$ *verts G* **unfolding** *tips-def*
using *Collect-subset* **by** *auto*

lemma *tips-tips*:
assumes $x \in$ *tips G*
shows *is-tip G x* **using** *tips-def CollectD assms*(*1*) **by** *metis*

1.2.2 Anticone

lemma (**in** *DAG*) *tips-anticone*:
assumes $a \in$ *tips G*

and $b \in$ *tips G*
and $a \neq b$
shows $a \in$ *anticone G b*
proof(*rule ccontr*)
assume $a \notin$ *anticone G b*
then have *k*: $(a \to^+ b \vee b \to^+ a \vee a = b)$ **using** *anticone.simps assms tips-def*
by *fastforce*
then have $\neg$ ($\forall x \in$*verts G*. $x \to^+ a$) $\vee$ $\neg$ ($\forall x \in$*verts G*. $x \to^+ b$) **using** *reachable1-in-verts*
assms(3) cycle-free
by (*metis*)
then have $\neg$ *is-tip G a* $\vee$ $\neg$ *is-tip G b* **using** *assms(3) is-tip.simps k*
by (*metis*)
then have $\neg$ $a \in$ *tips G* $\vee$ $\neg$ $b \in$ *tips G* **using** *tips-def CollectD* **by** *metis*
then show *False* **using** *assms* **by** *auto*
qed

lemma (**in** *DAG*) *anticone-in-verts*:
shows *anticone G a* $\subseteq$ *verts G* **using** *anticone.simps* **by** *auto*

lemma (**in** *DAG*) *anticon-finite*:
shows *finite* (*anticone G a*) **using** *anticone-in-verts* **by** *auto*

lemma (**in** *DAG*) *anticon-not-refl*:
shows $a \notin$ (*anticone G a*) **by** *auto*

1.2.3 Future Nodes

lemma (**in** *DAG*) *future-nodes-not-refl*:
assumes $a \in$ *verts G*
shows $a \notin$ *future-nodes G a*
using *cycle-free future-nodes.simps reachable-def* **by** *auto*

1.2.4 Past Nodes

lemma (**in** *DAG*) *past-nodes-not-refl*:
assumes $a \in$ *verts G*
shows $a \notin$ *past-nodes G a*
using *cycle-free past-nodes.simps reachable-def* **by** *auto*

lemma (**in** *DAG*) *past-nodes-verts*:
shows *past-nodes G a* $\subseteq$ *verts G*
using *past-nodes.simps reachable1-in-verts* **by** *auto*

lemma (**in** *DAG*) *past-nodes-refl-ex*:
assumes $a \in$ *verts G*
shows $a \in$ *past-nodes-refl G a*
using *past-nodes-refl.simps reachable-refl assms*
by *simp*

lemma (**in** *DAG*) *past-nodes-refl-verts*:
shows *past-nodes-refl G a* $\subseteq$ *verts G*
using *past-nodes.simps reachable-in-verts* **by** *auto*

lemma (**in** *DAG*) *finite-past*: *finite* (*past-nodes G a*)
by (*metis finite-verts rev-finite-subset past-nodes-verts*)

lemma (**in** *DAG*) *future-nodes-verts*:
shows *future-nodes G a* $\subseteq$ *verts G*
using *future-nodes.simps reachable1-in-verts* **by** *auto*

lemma (**in** *DAG*) *finite-future*: *finite* (*future-nodes G a*)
by (*metis finite-verts rev-finite-subset future-nodes-verts*)

lemma (**in** *DAG*) *past-future-dis*[*simp*]: *past-nodes G a* $\cap$ *future-nodes G a* $= \{\}$
proof (*rule ccontr*)
assume $\neg$ *past-nodes G a* $\cap$ *future-nodes G a* $= \{\}$
then show *False*
using *past-nodes.simps future-nodes.simps unidirectional reachable1-reachable*
by *auto*
qed

1.2.5 Reduce Past

lemma (**in** *DAG*) *reduce-past-arcs*:
shows *arcs* (*reduce-past G a*) $\subseteq$ *arcs G*
using *induce-subgraph-arcs past-nodes.simps* **by** *auto*

lemma (**in** *DAG*) *reduce-past-arcs2*:
$e \in$ *arcs* (*reduce-past G a*) $\Longrightarrow e \in$ *arcs G*
using *reduce-past-arcs* **by** *auto*

lemma (**in** *DAG*) *reduce-past-induced-subgraph*:
shows *induced-subgraph* (*reduce-past G a*) *G*
using *induced-induce past-nodes-verts* **by** *auto*

lemma (**in** *DAG*) *reduce-past-path*:
assumes $u \rightarrow^+{}_{\textit{reduce-past } G\ a}\ v$
shows $u \rightarrow^+{}_G\ v$
using *assms*
proof *induct*
case *base* **then show** *?case*
using *dominates-induce-subgraphD r-into-trancl′ reduce-past.simps*
by *metis*
next case (*step u v*) **show** *?case*
using *dominates-induce-subgraphD reachable1-reachable-trans reachable-adjI*
reduce-past.simps step.hyps(*2*) *step.hyps*(*3*) **by** *metis*

qed

lemma (**in** *DAG*) *reduce-past-path2*:
  assumes $u \to^{+}{}_{G}\ v$
  and $u \in$ *past-nodes G a*
  and $v \in$ *past-nodes G a*
  shows $u \to^{+}{}_{\textit{reduce-past } G\ a}\ v$
  using *assms*
proof(*induct u v*)
  case (*r-into-trancl u v*)
  then obtain *e* **where** *e-in*: *arc e* (*u*,*v*) **using** *arc-def DAG-axioms wf-digraph-def*
    by *auto*
  then have *e-in2*: $e \in$ *arcs* (*reduce-past G a*) **unfolding** *reduce-past.simps induce-subgraph-arcs*
    using *arcE r-into-trancl.prems*(*1*) *r-into-trancl.prems*(*2*) **by** *blast*
  then have *arc-to-ends* (*reduce-past G a*) *e* = (*u*,*v*) **unfolding** *reduce-past.simps* **using** *e-in*
  arcE arc-to-ends-def induce-subgraph-head induce-subgraph-tail
    by *metis*

  then have $u \to_{\textit{reduce-past } G\ a}\ v$ **using** *e-in2 wf-digraph.dominatesI DAG-axioms*
    by (*metis reduce-past.simps wellformed-induce-subgraph*)
  then show *?case* **by** *auto*
next
  case (*trancl-into-trancl a2 b c*)
  then have *b-in*: $b \in$ *past-nodes G a* **unfolding** *past-nodes.simps*
    by (*metis* (*mono-tags*, *lifting*) *adj-in-verts*(*1*) *mem-Collect-eq*
      reachable1-reachable reachable1-reachable-trans)
  then have *a2-re-b*: $a2 \to^{+}{}_{\textit{reduce-past } G\ a}\ b$ **using** *trancl-into-trancl* **by** *auto*
  then obtain *e* **where** *e-in*: *arc e* (*b*,*c*) **using** *trancl-into-trancl*
    arc-def DAG-axioms wf-digraph-def **by** *auto*
  then have *e-in2*: $e \in$ *arcs* (*reduce-past G a*) **unfolding** *reduce-past.simps induce-subgraph-arcs*
    using *arcE trancl-into-trancl*
    b-in **by** *blast*
  then have *arc-to-ends* (*reduce-past G a*) *e* = (*b*,*c*) **unfolding** *reduce-past.simps* **using** *e-in*
  arcE arc-to-ends-def induce-subgraph-head induce-subgraph-tail
    by *metis*
  then have $b \to_{\textit{reduce-past } G\ a}\ c$ **using** *e-in2 wf-digraph.dominatesI DAG-axioms*
    by (*metis reduce-past.simps wellformed-induce-subgraph*)
  then show *?case* **using** *a2-re-b*
    by (*metis trancl.trancl-into-trancl*)
qed

lemma (**in** *DAG*) *reduce-past-pathr*:
  assumes $u \to^{*}{}_{\textit{reduce-past } G\ a}\ v$
  shows $u \to^{*}{}_{G}\ v$

by (*meson assms induced-subgraph-altdef reachable-mono reduce-past-induced-subgraph*)

1.2.6 Reduce Past Reflexiv

lemma (**in** *DAG*) *reduce-past-refl-induced-subgraph*:
  shows *induced-subgraph* (*reduce-past-refl G a*) *G*
  using *induced-induce past-nodes-refl-verts* **by** *auto*

lemma (**in** *DAG*) *reduce-past-refl-arcs2*:
  $e \in$ *arcs* (*reduce-past-refl G a*) $\Longrightarrow$ $e \in$ *arcs G*
  using *reduce-past-arcs* **by** *auto*

lemma (**in** *DAG*) *reduce-past-refl-digraph*:
  assumes $a \in$ *verts G*
  shows *digraph* (*reduce-past-refl G a*)
  using *digraphI-induced reduce-past-refl-induced-subgraph reachable-mono* **by** *simp*

1.2.7 Reachability cases

lemma (**in** *DAG*) *reachable1-cases*:
  obtains (*nR*) $\neg\ a \rightarrow^+ b \wedge \neg\ b \rightarrow^+ a \wedge a \neq b$
  | (*one*) $a \rightarrow^+ b$
  | (*two*) $b \rightarrow^+ a$
  | (*eq*) $a = b$
  using *reachable-neq-reachable1 DAG-axioms*
  by *metis*

lemma (**in** *DAG*) *verts-comp*:
  assumes $x \in$ *tips G*
  shows *verts G* $= \{x\} \cup$ (*anticone G x*) $\cup$ (*verts* (*reduce-past G x*))
proof
  show *verts G* $\subseteq \{x\} \cup$ *anticone G x* $\cup$ *verts* (*reduce-past G x*)
  proof(*rule subsetI*)
    fix *xa*
    assume *in-V*: *xa* $\in$ *verts G*
    then show *xa* $\in \{x\} \cup$ *anticone G x* $\cup$ *verts* (*reduce-past G x*)
    proof(*cases x xa rule*: *reachable1-cases*)
      case *nR*
      then show *?thesis* **using** *anticone.simps in-V* **by** *auto*
      next
        case *one*
     then show *?thesis* **using** *reduce-past.simps induce-subgraph-verts past-nodes.simps in-V*
          by *auto*
      next
        case *two*
        have *is-tip G x* **using** *tips-tips assms*(*1*) **by** *simp*
        then have *False* **using** *tips-not-referenced two* **by** *auto*
        then show *?thesis* **by** *simp*
      next

case *eq*
then show *?thesis* **by** *auto*
qed
qed
next
show $\{x\} \cup$ *anticone* $G\ x \cup$ *verts* (*reduce-past* $G\ x$) $\subseteq$ *verts* G **using** *digraph.tips-in-verts*
digraph-axioms anticone-in-verts reduce-past-induced-subgraph induced-subgraph-def subgraph-def assms **by** *auto*
qed

lemma (**in** *DAG*) *verts-comp-dis*:
shows $\{x\} \cap$ (*anticone* $G\ x$) $= \{\}$
and $\{x\} \cap$ (*verts* (*reduce-past* $G\ x$)) $= \{\}$
and *anticone* $G\ x \cap$ (*verts* (*reduce-past* $G\ x$)) $= \{\}$
proof(*simp-all*, *simp add*: *cycle-free*, *safe*) **qed**

lemma (**in** *DAG*) *verts-size-comp*:
assumes $x \in$ *tips* G
shows *card* (*verts* G) $=$ *1* $+$ *card* (*anticone* $G\ x$) $+$ *card* (*verts* (*reduce-past* $G\ x$))
proof −
have *f1*: *finite* (*verts* G) **using** *finite-verts* **by** *simp*
have *f2*: *finite* $\{x\}$ **by** *auto*
have *f3*: *finite* (*anticone* $G\ x$) **using** *anticone.simps* **by** *auto*
have *f4*: *finite* (*verts* (*reduce-past* $G\ x$)) **by** *auto*
have *c1*: *card* $\{x\}$ $+$ *card* (*anticone* $G\ x$) $=$ *card* ($\{x\} \cup$ (*anticone* $G\ x$)) **using** *card-Un-disjoint*
verts-comp-dis **by** *auto*
have ($\{x\} \cup$ (*anticone* $G\ x$)) $\cap$ *verts* (*reduce-past* $G\ x$) $= \{\}$ **using** *verts-comp-dis* **by** *auto*
then have *card* ($\{x\} \cup$ (*anticone* $G\ x$) $\cup$ *verts* (*reduce-past* $G\ x$))
$=$ *card* $\{x\}$ $+$ *card* (*anticone* $G\ x$) $+$ *card* (*verts* (*reduce-past* $G\ x$))
using *card-Un-disjoint*
by (*metis c1 f2 f3 f4 finite-UnI*)
moreover have *card* (*verts* G) $=$ *card* ($\{x\} \cup$ (*anticone* $G\ x$) $\cup$ *verts* (*reduce-past* $G\ x$))
using *assms verts-comp* **by** *auto*
moreover have *card* $\{x\} =$ *1* **by** *simp*
ultimately show *?thesis* **using** *assms verts-comp*
by *presburger*
qed

end

theory *DigraphUtils*

imports *Main Graph-Theory.Graph-Theory*
begin

2 Digraph Utilities

lemma *graph-equality*:
assumes *digraph G* ∧ *digraph C*
assumes *verts G* = *verts C* ∧ *arcs G* = *arcs C* ∧ *head G* = *head C* ∧ *tail G* = *tail C*
shows *G* = *C*
by (*simp add*: *assms*(*2*))

lemma (**in** *digraph*) *del-vert-not-in-graph*:
assumes *b* ∉ *verts G*
shows (*pre-digraph.del-vert G b*) = *G*
proof −
have *v*: *verts* (*pre-digraph.del-vert G b*) = *verts G*
using *assms*(*1*)
by (*simp add*: *pre-digraph.verts-del-vert*)
have ∀ *e* ∈ *arcs G*. *tail G e* ≠ *b* ∧ *head G e* ≠ *b* **using** *digraph-axioms assms digraph.axioms*(*2*) *loopfree-digraph.axioms*(*1*)
by *auto*
then have *arcs G* ⊆ *arcs* (*pre-digraph.del-vert G b*)
using *assms*
by (*simp add*: *pre-digraph.arcs-del-vert subsetI*)
then have *e*: *arcs G* = *arcs* (*pre-digraph.del-vert G b*)
by (*simp add*: *pre-digraph.arcs-del-vert subset-antisym*)
then show *?thesis* **using** *v* **by** (*simp add*: *pre-digraph.del-vert-simps*)
qed

lemma *del-arc-subgraph*:
assumes *subgraph H G*
assumes *digraph G* ∧ *digraph H*
shows *subgraph* (*pre-digraph.del-arc H e2*) (*pre-digraph.del-arc G e2*)
using *subgraph-def pre-digraph.del-arc-simps Diff-iff*
proof −
have *f1*: ∀ *p pa*. *subgraph p pa* = ((*verts p*::'*a set*) ⊆ *verts pa* ∧ (*arcs p*::'*b set*) ⊆ *arcs pa* ∧
wf-digraph pa ∧ *wf-digraph p* ∧ *compatible pa p*)
using *subgraph-def* **by** *blast*
have *arcs H* − {*e2*} ⊆ *arcs G* − {*e2*} **using** *assms*(*1*)
by *auto*
then show *?thesis*
unfolding *subgraph-def*
using *f1 assms*(*1*) **by** (*simp add*: *compatible-def pre-digraph.del-arc-simps wf-digraph.wf-digraph-del-arc*)
qed

lemma *graph-nat-induct*[*consumes 0*, *case-names base step*]:
assumes

cases: $\bigwedge V.\ (digraph\ V \Longrightarrow card\ (verts\ V) = 0 \Longrightarrow P\ V)$
$\bigwedge W\ c.\ (\bigwedge V.\ (digraph\ V \Longrightarrow card\ (verts\ V) = c \Longrightarrow P\ V))$
$\Longrightarrow (digraph\ W \Longrightarrow card\ (verts\ W) = (Suc\ c) \Longrightarrow P\ W)$
shows $\bigwedge Z.\ digraph\ Z \Longrightarrow P\ Z$
proof −
fix Z:: $('a,'b)$ *pre-digraph*
assume *major*: *digraph Z*
then show *P Z*
proof (*induction card* (*verts Z*) *arbitrary*: *Z*)
case *0*
then show *?case*
by (*simp add*: *local.cases*(*1*) *major*)
next
case *su*: (*Suc x*)
assume $(\bigwedge Z.\ x = card\ (verts\ Z) \Longrightarrow digraph\ Z \Longrightarrow P\ Z)$
show *?case*
by (*metis local.cases*(*2*) *su.hyps*(*1*) *su.hyps*(*2*) *su.prems*)
qed
qed
end

theory *blockDAG*
imports *DAGs DigraphUtils*
begin

3 blockDAGs

locale *blockDAG* = *DAG* +
assumes *genesis*: $\exists p \in verts\ G.\ \forall r.\ r \in verts\ G \longrightarrow (r \rightarrow^{+}{}_{G}\ p \vee r = p)$
and *only-new*: $\forall e.\ (u \rightarrow^{+}{}_{(del\text{-}arc\ e)}\ v) \longrightarrow \neg\ arc\ e\ (u,v)$

3.1 Functions and Definitions

fun (**in** *blockDAG*) *is-genesis-node* :: $'a \Rightarrow bool$ **where**
is-genesis-node $v = ((v \in verts\ G) \wedge (ALL\ x.\ (x \in verts\ G) \longrightarrow x \rightarrow^{*}{}_{G}\ v))$

definition (**in** *blockDAG*) *genesis-node*:: $'a$
where *genesis-node* = (*THE x. is-genesis-node x*)

3.2 Lemmas

lemma *subs*:
assumes *blockDAG G*
shows $DAG\ G \wedge digraph\ G \wedge fin\text{-}digraph\ G \wedge wf\text{-}digraph\ G$

using *assms blockDAG-def DAG-def digraph-def fin-digraph-def* **by** *blast*

3.2.1 Genesis

lemma (**in** *blockDAG*) *genesisAlt* :
$(is\text{-}genesis\text{-}node\ a) \longleftrightarrow ((a \in verts\ G) \wedge (\forall r.\ (r \in verts\ G) \longrightarrow r \rightarrow^* a))$
by *simp*

lemma (**in** *blockDAG*) *genesis-existAlt*:
$\exists a.\ is\text{-}genesis\text{-}node\ a$
using *genesis genesisAlt*
by (*metis reachable1-reachable reachable-refl*)

lemma (**in** *blockDAG*) *unique-genesis*: $is\text{-}genesis\text{-}node\ a \wedge is\text{-}genesis\text{-}node\ b \longrightarrow a = b$
using *genesisAlt reachable-trans cycle-free*
reachable-refl reachable-reachable1-trans reachable-neq-reachable1
by (*metis* (*full-types*))

lemma (**in** *blockDAG*) *genesis-unique-exists*:
$\exists!a.\ is\text{-}genesis\text{-}node\ a$
using *genesis-existAlt unique-genesis* **by** *auto*

lemma (**in** *blockDAG*) *genesis-in-verts*:
$genesis\text{-}node \in verts\ G$
using *is-genesis-node.simps genesis-node-def genesis-existAlt the1I2 genesis-unique-exists*
by *metis*

3.2.2 Tips

lemma (**in** *blockDAG*) *tips-exist*:
$\exists x.\ is\text{-}tip\ G\ x$
unfolding *is-tip.simps*
proof (*rule ccontr*)
assume $\nexists x.\ x \in verts\ G \wedge (\forall xa \in verts\ G.\ (xa, x) \notin (arcs\text{-}ends\ G)^+)$
then have *contr*: $\forall x.\ x \in verts\ G \longrightarrow (\exists y.\ y \rightarrow^+ x)$
by *auto*
have $\forall\ x\ y.\ y \rightarrow^+ x \longrightarrow \{z.\ x \rightarrow^+ z\} \subseteq \{z.\ y \rightarrow^+ z\}$
using *Collect-mono trancl-trans*
by *metis*
then have *sub*: $\forall\ x\ y.\ y \rightarrow^+ x \longrightarrow \{z.\ x \rightarrow^+ z\} \subset \{z.\ y \rightarrow^+ z\}$
using *cycle-free* **by** *auto*
have *part*: $\forall\ x.\ \{z.\ x \rightarrow^+ z\} \subseteq verts\ G$
using *reachable1-in-verts* **by** *auto*
then have *fin*: $\forall\ x.\ finite\ \{z.\ x \rightarrow^+ z\}$
using *finite-verts finite-subset*
by *metis*
then have *trans*: $\forall\ x\ y.\ y \rightarrow^+ x \longrightarrow card\ \{z.\ x \rightarrow^+ z\} < card\ \{z.\ y \rightarrow^+ z\}$
using *sub psubset-card-mono* **by** *metis*
then have *inf*: $\forall y \in verts\ G.\ \exists x.\ card\ \{z.\ x \rightarrow^+ z\} > card\ \{z.\ y \rightarrow^+ z\}$

```
    using fin contr genesis
      reachable1-in-verts(1)
    by (metis (mono-tags, lifting))
  have all: ∀ k. ∃ x ∈ verts G. card {z. x →⁺ z} > k
  proof
    fix k
    show ∃ x ∈ verts G. k < card {z. x →⁺ z}
    proof(induct k)
      case 0
      then show ?case
        using inf neq0-conv
        by (metis contr genesis-in-verts local.trans reachable1-in-verts(1))
    next
      case (Suc k)
      then show ?case
        using Suc-lessI inf
        by (metis contr local.trans reachable1-in-verts(1))
    qed
  qed
  then have less: ∃ x ∈ verts G. card (verts G) < card {z. x →⁺ z} by simp
  also
  have ∀ x. card {z. x →⁺ z} ≤ card (verts G)
    using fin part finite-verts not-le
    by (simp add: card-mono)
  then show False
    using less not-le by auto
qed

lemma (in blockDAG) tips-not-empty:
  shows tips G ≠ {}
proof(rule ccontr)
  assume as1: ¬ tips G ≠ {}
  obtain t where t-in: is-tip G t using tips-exist by auto
  then have t-inV: t ∈ verts G by auto
  then have t ∈ tips G using tips-def CollectI t-in by metis
  then show False using as1 by auto
qed

lemma (in blockDAG) tips-unequal-gen:
  assumes card( verts G) > 1
  and is-tip G p
  shows ¬ is-genesis-node p
proof (rule ccontr)
  assume as: ¬ ¬ is-genesis-node p
  have b1: 1 < card (verts G) using assms by linarith
   then have 0 < card ((verts G) − {p}) using card-Suc-Diff1 as finite-verts b1
by auto
  then have ((verts G) − {p}) ≠ {} using card-gt-0-iff by blast
  then obtain y where y-def:y ∈ (verts G) − {p} by auto
```

then have *uneq*: $y \neq p$ **by** *auto*
then have *reachable1 G y p* **using** *is-genesis-node.simps as reachable-neq-reachable1 Diff-iff y-def*
by *metis*
then have $\neg$ *is-tip G p*
by (*meson is-tip.elims*(*2*) *reachable1-in-verts*(*1*))
then show *False* **using** *assms* **by** *simp*
qed

lemma (**in** *blockDAG*) *tips-unequal-gen-exist*:
assumes *card*(*verts G*) > *1*
shows $\exists$ *p*. $p \in$ *verts G* $\wedge$ *is-tip G p* $\wedge$ $\neg$*is-genesis-node p*
proof −
have *b1*: *1* < *card* (*verts G*) **using** *assms* **by** *linarith*
obtain *x* **where** *x-in*: $x \in$ (*verts G*) $\wedge$ *is-genesis-node x*
using *genesis genesisAlt genesis-node-def* **by** *blast*
then have *0* < *card* ((*verts G*) − {*x*}) **using** *card-Suc-Diff1 x-in finite-verts b1* **by** *auto*
then have ((*verts G*) − {*x*}) $\neq$ {} **using** *card-gt-0-iff* **by** *blast*
then obtain *y* **where** *y-def*:$y \in$ (*verts G*) − {*x*} **by** *auto*
then have *uneq*: $y \neq x$ **by** *auto*
have *y-in*: $y \in$ (*verts G*) **using** *y-def* **by** *simp*
then have *reachable1 G y x* **using** *is-genesis-node.simps x-in reachable-neq-reachable1 uneq* **by** *simp*
then have $\neg$ *is-tip G x*
by (*meson is-tip.elims*(*2*) *y-in*)
then obtain *z* **where** *z-def*: $z \in$ (*verts G*) − {*x*} $\wedge$ *is-tip G z* **using** *tips-exist is-tip.simps* **by** *auto*
then have *uneq*: $z \neq x$ **by** *auto*
have *z-in*: $z \in$ *verts G* **using** *z-def* **by** *simp*
have $\neg$ *is-genesis-node z*
proof (*rule ccontr*, *safe*)
assume *is-genesis-node z*
then have $x = z$ **using** *unique-genesis x-in* **by** *auto*
then show *False* **using** *uneq* **by** *simp*
qed
then show *?thesis* **using** *z-def* **by** *auto*
qed

lemma (**in** *blockDAG*) *del-tips-bDAG*:
assumes *is-tip G t*
and $\neg$*is-genesis-node t*
shows *blockDAG* (*del-vert t*)
unfolding *blockDAG-def blockDAG-axioms-def*
proof *safe*
show *DAG*(*del-vert t*)
using *del-tips-dag assms* **by** *simp*

```
next
  fix u v e
  assume wf-digraph.arc (del-vert t) e (u, v)
  then have arc: arc e (u,v) using del-vert-simps wf-digraph.arc-def arc-def
    by (metis (no-types, lifting) mem-Collect-eq wf-digraph-del-vert)
  assume u →^+_{pre-digraph.del-arc (del-vert t) e} v
  then have path: u →^+_{del-arc e} v
    using  del-arc-subgraph subgraph-del-vert digraph-axioms
        digraph-subgraph
    by (metis arcs-ends-mono trancl-mono)
  show False using arc path only-new by simp
next
  obtain g where gen: is-genesis-node g using genesisAlt genesis by auto
  then have genp: g ∈ verts (del-vert t)
    using assms(2) genesis del-vert-simps by auto
  have (∀ r. r ∈ verts (del-vert t) ⟶ r →^*_{del-vert t} g)
  proof safe
  fix r
  assume in-del: r ∈ verts (del-vert t)
  then obtain p where path: awalk r p g
    using reachable-awalk is-genesis-node.simps del-vert-simps gen by auto
  have no-head: t ∉ (set ( map (λs. (head G s)) p))
  proof (rule ccontr)
    assume ¬ t ∉ (set ( map (λs. (head G s)) p))
    then have as: t ∈ (set ( map (λs. (head  G s)) p))
    by auto
    then obtain e where tl: t = (head G e) ∧ e ∈ arcs G
      using wf-digraph-def awalk-def path by auto
    then obtain u where hd: u = (tail G e) ∧ u ∈ verts G
      using wf-digraph-def tl by auto
    have t ∈ verts G
      using assms(1) is-tip.simps by auto
    then have arc-to-ends G e = (u, t) using tl
      by (simp add: arc-to-ends-def hd)
    then have reachable1 G u t
      using dominatesI tl by blast
    then show False
      using is-tip.simps assms(1)
      hd by auto
  qed
  have neither: r ≠ t ∧ g ≠ t
      using del-vert-def assms(2) gen in-del by auto
  have no-tail: t ∉ (set ( map (tail G) p))
  proof(rule ccontr)
    assume as2: ¬ t ∉ set (map (tail G) p)
    then have tl2: t ∈ set (map (tail G) p) by auto
    then have t ∈ set (map (head G) p)
    proof (induct rule: cas.induct)
      case (1 u v)
```

```
      then have v ∉ set (map (tail G) []) by auto
      then show v ∈ set (map (tail G) []) ⟹ v ∈ set (map (head G) [])
        by auto
    next
      case (2 u e es v)
      then show ?case
        using set-awalk-verts-not-Nil-cas neither awalk-def cas.simps(2) path
        by (metis UnCI tl2 awalk-verts-conv'
            cas-simp list.simps(8) no-head set-ConsD)
    qed
    then show False using no-head by auto
  qed
  have pre-digraph.awalk (del-vert t) r p g
    unfolding pre-digraph.awalk-def
  proof safe
    show r ∈ verts (del-vert t) using in-del by simp
  next
    fix x
    assume as3: x ∈ set p
    then have ht: head G x ≠ t ∧ tail G x ≠ t
      using no-head no-tail by auto
    have  x ∈ arcs G
      using awalk-def path subsetD as3 by auto
    then show x ∈ arcs (del-vert t) using del-vert-simps(2) ht by auto
  next
    have pre-digraph.cas G r p g using path by auto
    then show pre-digraph.cas (del-vert t) r p g
    proof(induct p arbitrary:r)
      case Nil
      then have r = g using awalk-def cas.simps by auto
      then show ?case using pre-digraph.cas.simps(1)
        by (metis)
    next
      case (Cons a p)
      assume pre: ⋀r. (cas r p g ⟹ pre-digraph.cas (del-vert t) r p g)
      and one: cas r (a # p) g
      then have two: cas (head G a) p g
        using awalk-def  by auto
      then have t: tail (del-vert t) a = r
        using one cas.simps awalk-def del-vert-simps(3) by auto
      then show ?case
        unfolding pre-digraph.cas.simps(2) t
        using pre two del-vert-simps(4) by auto
    qed
  qed
  then show r →*del-vert t g by (meson wf-digraph.reachable-awalkI
  del-tips-dag assms(1) DAG-def digraph-def fin-digraph-def)
qed
  then show  ∃p ∈ verts (del-vert t) .
```

$(\forall r.\ r \in$ *verts* (*del-vert t*) $\longrightarrow (r \rightarrow^{+}{}_{\textit{del-vert } t}\ p \vee r = p))$
using *gen genp*
by (*metis reachable-rtranclI rtranclD*)
qed

lemma (**in** *blockDAG*) *tips-cases* [*consumes 2*, *case-names ma past nma*]:
assumes $p \in$ *tips G*
and $x \in$ *verts G*
obtains (*ma*) $x = p$
| (*past*) $x \in$ *past-nodes G p*
| (*nma*) $x \in$ *anticone G p*
proof –
consider (*eq*)$x = p$ | (*neq*) $\neg x = p$ **by** *auto*
then show *?thesis*
proof(*cases*)
case *eq*
then show *thesis* **using** *eq ma* **by** *simp*
next
case *neq*
consider (*in-p*)$x \in$ *past-nodes G p* | (*nin-p*)$x \notin$ *past-nodes G p* **by** *auto*
then show *?thesis*
proof(*cases*)
case *in-p*
then show *?thesis* **using** *past* **by** *auto*
next
case *nin-p*
then have *nn*: $\neg\ p \rightarrow^{+}{}_{G}\ x$ **using** *nin-p past-nodes.simps assms*(*2*) **by** *auto*
have $\neg\ x \rightarrow^{+}{}_{G}\ p$ **using** *is-tip.simps assms tips-def CollectD* **by** *metis*
then have $x \in$ *anticone G p* **using** *anticone.simps neq nn assms*(*2*) **by** *auto*
then show *?thesis* **using** *nma* **by** *auto*
qed
qed
qed

3.3 Future Nodes

lemma (**in** *blockDAG*) *future-nodes-ex*:
assumes $a \in$ *verts G*
shows $a \notin$ *future-nodes G a*
using *cycle-free future-nodes.simps reachable-def* **by** *auto*

3.3.1 Reduce Past

lemma (**in** *blockDAG*) *reduce-past-not-empty*:
assumes $a \in$ *verts G*
and $\neg$*is-genesis-node a*
shows (*verts* (*reduce-past G a*)) $\neq \{\}$
proof –
obtain *g*

where *gen*: *is-genesis-node g* **using** *genesis-existAlt* **by** *auto*
have *ex*: $g \in$ *verts* (*reduce-past G a*) **using** *reduce-past.simps past-nodes.simps genesisAlt reachable-neq-reachable1 reachable-reachable1-trans gen assms*(*1*) *assms*(*2*)
by *auto*
then show (*verts* (*reduce-past G a*)) $\neq \{\}$ **using** *ex* **by** *auto*
qed

lemma (**in** *blockDAG*) *reduce-less*:
assumes $a \in$ *verts G*
shows *card* (*verts* (*reduce-past G a*)) $<$ *card* (*verts G*)
proof –
have *past-nodes G a* $\subset$ *verts G*
using *assms*(*1*) *past-nodes-not-refl past-nodes-verts* **by** *blast*
then show *?thesis*
by (*simp add*: *psubset-card-mono*)
qed

lemma (**in** *blockDAG*) *reduce-past-dagbased*:
assumes $a \in$ *verts G*
and $\neg$*is-genesis-node a*
shows *blockDAG* (*reduce-past G a*)
unfolding *blockDAG-def DAG-def blockDAG-def*

proof *safe*
show *digraph* (*reduce-past G a*)
using *digraphI-induced reduce-past-induced-subgraph* **by** *auto*
next
show *DAG-axioms* (*reduce-past G a*)
unfolding *DAG-axioms-def*
using *cycle-free reduce-past-path* **by** *metis*
next
show *blockDAG-axioms* (*reduce-past G a*)
unfolding *blockDAG-axioms-def*
proof *safe*
fix *u v e*
assume *arc*: *wf-digraph.arc* (*reduce-past G a*) *e* (*u*, *v*)
then show $u \rightarrow^{+}_{\textit{pre-digraph.del-arc (reduce-past G a) e}} v \Longrightarrow \textit{False}$
proof –
assume *e-in*: (*wf-digraph.arc* (*reduce-past G a*) *e* (*u*, *v*))
then have (*wf-digraph.arc G e* (*u*, *v*))
using *assms reduce-past-arcs2 induced-subgraph-def arc-def*
proof –
have *wf-digraph* (*reduce-past G a*)
using *reduce-past.simps subgraph-def subgraph-refl wf-digraph.wellformed-induce-subgraph*
by *metis*
then have $e \in$ *arcs* (*reduce-past G a*) $\wedge$ *tail* (*reduce-past G a*) *e* $=$ *u*

$\wedge$ *head* (*reduce-past G a*) *e* = *v*
using *arc wf-digraph.arcE*
by *metis*
then show *?thesis*
using *arc-def reduce-past.simps* **by** *auto*
qed
then have $\neg\ u \rightarrow^{+}{}_{del\text{-}arc\ e}\ v$
using *only-new* **by** *auto*
then show $u \rightarrow^{+}{}_{pre\text{-}digraph.del\text{-}arc\ (reduce\text{-}past\ G\ a)\ e}\ v \Longrightarrow False$
using *DAG.past-nodes-verts reduce-past.simps blockDAG-axioms subs del-arc-subgraph digraph.digraph-subgraph digraph-axioms subgraph-induce-subgraphI*
by (*metis arcs-ends-mono trancl-mono*)
qed
next
obtain *p* **where** *gen*: *is-genesis-node p* **using** *genesis-existAlt* **by** *auto*
have *pe*: $p \in$ *verts* (*reduce-past G a*) $\wedge$ ($\forall r.\ r \in$ *verts* (*reduce-past G a*) $\longrightarrow$ $r \rightarrow^{*}{}_{reduce\text{-}past\ G\ a}\ p$)
proof
show $p \in$ *verts* (*reduce-past G a*) **using** *genesisAlt induce-reachable-preserves-paths reduce-past.simps past-nodes.simps reachable1-reachable induce-subgraph-verts assms*(*1*)
assms(*2*) *gen mem-Collect-eq reachable-neq-reachable1*
by (*metis* (*no-types*, *lifting*))

next
show $\forall r.\ r \in$ *verts* (*reduce-past G a*) $\longrightarrow r \rightarrow^{*}{}_{reduce\text{-}past\ G\ a}\ p$
proof *safe*
fix *r a*
assume *in-past*: $r \in$ *verts* (*reduce-past G a*)
then have *con*: $r \rightarrow^{*} p$ **using** *gen genesisAlt past-nodes-verts* **by** *auto*
then show $r \rightarrow^{*}{}_{reduce\text{-}past\ G\ a}\ p$
proof –
have *f1*: $r \in$ *verts G* $\wedge\ a \rightarrow^{+} r$
using *in-past past-nodes-verts* **by** *force*
obtain *aaa* :: *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *'a* **where**
f2: $\forall x0\ x1.\ (\exists v2.\ v2 \in x1 \wedge v2 \notin x0) = (aaa\ x0\ x1 \in x1 \wedge aaa\ x0\ x1 \notin x0)$
by *moura*
have $r \rightarrow^{*}$ *aaa* (*past-nodes G a*) (*Collect* (*reachable G r*))
$\longrightarrow a \rightarrow^{+}$ *aaa* (*past-nodes G a*) (*Collect* (*reachable G r*))
using *f1* **by** (*meson reachable1-reachable-trans*)
then have *aaa* (*past-nodes G a*) (*Collect* (*reachable G r*)) $\notin$ *Collect* (*reachable G r*)
$\vee$ *aaa* (*past-nodes G a*) (*Collect* (*reachable G r*)) $\in$ *past-nodes G a*
by (*simp add*: *reachable-in-verts*(*2*))
then have *Collect* (*reachable G r*) $\subseteq$ *past-nodes G a*
using *f2* **by** (*meson subsetI*)

then show *?thesis*
using *con induce-reachable-preserves-paths reachable-induce-ss reduce-past.simps*
by (*metis* (*no-types*))
qed
qed
qed
show
$\exists\, p \in verts\ (reduce\text{-}past\ G\ a).\ (\forall\, r.\ r \in verts\ (reduce\text{-}past\ G\ a) \longrightarrow (r \rightarrow^{+}{}_{reduce\text{-}past\ G\ a}\ p \vee r = p))$
using *pe*
by (*metis reachable-rtranclI rtranclD*)
qed
qed

lemma (**in** *blockDAG*) *reduce-past-gen*:
assumes $\neg$*is-genesis-node a*
and $a \in verts\ G$
shows *blockDAG.is-genesis-node G b* $\Longrightarrow$ *blockDAG.is-genesis-node* (*reduce-past G a*) *b*
proof –
assume *gen*: *blockDAG.is-genesis-node G b*
have *une*: $b \neq a$ **using** *gen assms*(*1*) *genesis-unique-exists* **by** *auto*
have $a \rightarrow^{*} b$ **using** *gen assms*(*2*) **by** *simp*
then have $a \rightarrow^{+} b$
using *reachable-neq-reachable1 is-genesis-node.simps assms*(*2*) *une* **by** *auto*
then have $b \in (past\text{-}nodes\ G\ a)$ **using** *past-nodes.simps gen* **by** *auto*
then have *inv*: $b \in verts\ (reduce\text{-}past\ G\ a)$ **using** *reduce-past.simps induce-subgraph-verts*
by *auto*
have $\forall\, r.\ r \in verts\ (reduce\text{-}past\ G\ a) \longrightarrow r \rightarrow^{*}{}_{reduce\text{-}past\ G\ a}\ b$
proof *safe*
fix *r a*
assume *in-past*: $r \in verts\ (reduce\text{-}past\ G\ a)$
then have *con*: $r \rightarrow^{*} b$ **using** *gen genesisAlt past-nodes-verts* **by** *auto*
then show $r \rightarrow^{*}{}_{reduce\text{-}past\ G\ a}\ b$
proof –
have *f1*: $r \in verts\ G \wedge a \rightarrow^{+} r$
using *in-past past-nodes-verts* **by** *force*
obtain *aaa* :: *'a set* $\Rightarrow$ *'a set* $\Rightarrow$ *'a* **where**
f2: $\forall\, x0\ x1.\ (\exists\, v2.\ v2 \in x1 \wedge v2 \notin x0) = (aaa\ x0\ x1 \in x1 \wedge aaa\ x0\ x1 \notin x0)$
by *moura*
have $r \rightarrow^{*} aaa\ (past\text{-}nodes\ G\ a)\ (Collect\ (reachable\ G\ r))$
$\longrightarrow a \rightarrow^{+} aaa\ (past\text{-}nodes\ G\ a)\ (Collect\ (reachable\ G\ r))$
using *f1* **by** (*meson reachable1-reachable-trans*)
then have $aaa\ (past\text{-}nodes\ G\ a)\ (Collect\ (reachable\ G\ r)) \notin Collect\ (reachable\ G\ r)$

∨ *aaa* (*past-nodes G a*) (*Collect* (*reachable G r*)) ∈ *past-nodes G a*
by (*simp add*: *reachable-in-verts*(*2*))
then have *Collect* (*reachable G r*) ⊆ *past-nodes G a*
using *f2* **by** (*meson subsetI*)
then show *?thesis*
using *con* *induce-reachable-preserves-paths* *reachable-induce-ss* *reduce-past.simps*
by (*metis* (*no-types*))
qed
qed
then show *blockDAG.is-genesis-node* (*reduce-past G a*) *b* **using** *inv is-genesis-node.simps*
by (*metis assms*(*1*) *assms*(*2*) *blockDAG.is-genesis-node.elims*(*3*)
reduce-past-dagbased)
qed

lemma (**in** *blockDAG*) *reduce-past-gen-rev*:
assumes ¬*is-genesis-node a*
and *a* ∈ *verts G*
shows *blockDAG.is-genesis-node* (*reduce-past G a*) *b* ⟹ *blockDAG.is-genesis-node G b*
proof −
assume *as1*: *blockDAG.is-genesis-node* (*reduce-past G a*) *b*
have *bD*: *blockDAG* (*reduce-past G a*) **using** *assms reduce-past-dagbased blockDAG-axioms* **by** *simp*
obtain *gen* **where** *is-gen*: *is-genesis-node gen* **using** *genesis-unique-exists* **by** *auto*
then have *blockDAG.is-genesis-node* (*reduce-past G a*) *gen* **using** *reduce-past-gen assms* **by** *auto*
then have *gen* = *b* **using** *as1 blockDAG.unique-genesis bD* **by** *metis*
then show *blockDAG.is-genesis-node* (*reduce-past G a*) *b* ⟹ *blockDAG.is-genesis-node G b*
using *is-gen* **by** *auto*
qed

lemma (**in** *blockDAG*) *reduce-past-gen-eq*:
assumes ¬*is-genesis-node a*
and *a* ∈ *verts G*
shows *blockDAG.is-genesis-node* (*reduce-past G a*) *b* = *blockDAG.is-genesis-node G b*
using *reduce-past-gen reduce-past-gen-rev assms assms* **by** *metis*

3.3.2 Reduce Past Reflexiv

lemma (**in** *blockDAG*) *reduce-past-refl-induced-subgraph*:
shows *induced-subgraph* (*reduce-past-refl G a*) *G*
using *induced-induce past-nodes-refl-verts* **by** *auto*

```
lemma (in blockDAG) reduce-past-refl-arcs2:
  e ∈ arcs (reduce-past-refl G a) ⟹ e ∈ arcs G
  using reduce-past-arcs by auto

lemma (in blockDAG) reduce-past-refl-digraph:
  assumes a ∈ verts G
  shows digraph (reduce-past-refl G a)
  using digraphI-induced reduce-past-refl-induced-subgraph reachable-mono by simp

lemma (in blockDAG) reduce-past-refl-dagbased:
  assumes a ∈ verts G
  shows blockDAG (reduce-past-refl G a)
  unfolding blockDAG-def DAG-def
proof safe
  show digraph (reduce-past-refl G a)
    using reduce-past-refl-digraph assms(1) by simp
next
  show DAG-axioms (reduce-past-refl G a)
    unfolding DAG-axioms-def
    using cycle-free reduce-past-refl-induced-subgraph reachable-mono
    by (meson arcs-ends-mono induced-subgraph-altdef trancl-mono)
next
  show blockDAG-axioms (reduce-past-refl G a)
    unfolding blockDAG-axioms
  proof
    fix u v
    show ∀ e. u →⁺pre-digraph.del-arc (reduce-past-refl G a) e v
        ⟶ ¬ wf-digraph.arc (reduce-past-refl G a) e (u, v)
    proof safe
      fix e
      assume a: wf-digraph.arc (reduce-past-refl G a) e (u, v)
      and b: u →⁺pre-digraph.del-arc (reduce-past-refl G a) e v
      have edge: wf-digraph.arc G e (u, v)
          using assms reduce-past-arcs2 induced-subgraph-def arc-def
        proof -
          have wf-digraph (reduce-past-refl G a)
            using reduce-past-refl-digraph digraph-def by auto
           then have e ∈ arcs (reduce-past-refl G a) ∧ tail (reduce-past-refl G a) e
= u
                   ∧ head (reduce-past-refl G a) e = v
            using wf-digraph.arcE arc-def a
            by (metis (no-types))
          then show arc e (u, v)
            using arc-def reduce-past-refl.simps by auto
        qed
      have u →⁺pre-digraph.del-arc G e v
        using a b reduce-past-refl-digraph del-arc-subgraph digraph-axioms
         digraphI-induced past-nodes-refl-verts reduce-past-refl.simps
         reduce-past-refl-induced-subgraph subgraph-induce-subgraphI arcs-ends-mono
```

trancl-mono
    by *metis*
  then show *False*
    using *edge only-new* **by** *simp*
  qed
next
    obtain *p* **where** *gen*: *is-genesis-node p* **using** *genesis-existAlt* **by** *auto*
    have *pe*: $p \in$ *verts* (*reduce-past-refl G a*)
    using *genesisAlt induce-reachable-preserves-paths*
    reduce-past.simps past-nodes.simps reachable1-reachable induce-subgraph-verts
      gen mem-Collect-eq reachable-neq-reachable1
      assms **by** *force*
  have *reaches*: ($\forall r.\ r \in$ *verts* (*reduce-past-refl G a*) $\longrightarrow$
      ($r \rightarrow^{+}{}_{\textit{reduce-past-refl } G\ a}\ p \vee r = p$))
    proof *safe*
      fix *r*
      assume *in-past*: $r \in$ *verts* (*reduce-past-refl G a*)
      assume *une*: $r \neq p$
      then have *con*: $r \rightarrow^{*} p$ **using** *gen genesisAlt reachable-in-verts*
      reachable1-reachable
        by (*metis in-past induce-subgraph-verts*
          past-nodes-refl-verts reduce-past-refl.simps subsetD)
      have $a \rightarrow^{*} r$ **using** *in-past* **by** *auto*
      then have *reach*: $r \rightarrow^{*}{}_{G \restriction \{w.\ a \rightarrow^{*} w\}}\ p$
      proof(*induction*)
        case *base*
        then show *?case*
          using *con induce-reachable-preserves-paths*
          by (*metis*)
      next
        case (*step x y*)
        then show *?case*
        proof −
          have *Collect* (*reachable G y*) $\subseteq$ *Collect* (*reachable G x*)
            using *adj-reachable-trans step.hyps*(*1*) **by** *force*
          then show *?thesis*
            using *reachable-induce-ss step.IH reachable-neq-reachable1*
            by *metis*
        qed
      qed
      then show $r \rightarrow^{+}{}_{\textit{reduce-past-refl } G\ a}\ p$ **unfolding** *reduce-past-refl.simps*
    past-nodes-refl.simps **using** *reachable-in-verts une wf-digraph.reachable-neq-reachable1*
      by (*metis* (*mono-tags*, *lifting*) *Collect-cong wellformed-induce-subgraph*)
    qed
    then show $\exists p \in$ *verts* (*reduce-past-refl G a*). ($\forall r.\ r \in$ *verts* (*reduce-past-refl*
G a)
    $\longrightarrow$ ($r \rightarrow^{+}{}_{\textit{reduce-past-refl } G\ a}\ p \vee r = p$)) **unfolding** *blockDAG-axioms-def*
      using *pe reaches* **by** *auto*
    qed

qed

3.3.3 Genesis Graph

definition (**in** *blockDAG*) *gen-graph*::(*'a*,*'b*) *pre-digraph* **where**
gen-graph = *induce-subgraph G* {*blockDAG.genesis-node G*}

lemma (**in** *blockDAG*) *gen-gen* :*verts* (*gen-graph*) = {*genesis-node*}
unfolding *genesis-node-def gen-graph-def* **by** *simp*

lemma (**in** *blockDAG*) *gen-graph-one*: *card* (*verts gen-graph*) = *1* **using** *gen-gen*
by *simp*

lemma (**in** *blockDAG*) *gen-graph-digraph*:
digraph gen-graph
using *digraphI-induced induced-induce gen-graph-def*
genesis-in-verts **by** *simp*

lemma (**in** *blockDAG*) *gen-graph-empty-arcs*:
arcs gen-graph = {}
proof(*rule ccontr*)
assume ¬ *arcs gen-graph* = {}
then have *ex*: ∃ *a*. *a* ∈ (*arcs gen-graph*)
by *blast*
also have ∀ *a*. *a* ∈ (*arcs gen-graph*)⟶ *tail G a* = *head G a*
proof *safe*
fix *a*
assume *a* ∈ *arcs gen-graph*
then show *tail G a* = *head G a*
using *digraph-def induced-subgraph-def induce-subgraph-verts*
induced-induce gen-graph-def **by** *simp*
qed
then show *False*
using *digraph-def ex gen-graph-def gen-graph-digraph induce-subgraph-head*
induce-subgraph-tail
loopfree-digraph.no-loops
by *metis*
qed

lemma (**in** *blockDAG*) *gen-graph-sound*:
blockDAG (*gen-graph*)
unfolding *blockDAG-def DAG-def blockDAG-axioms-def*
proof *safe*
show *digraph gen-graph* **using** *gen-graph-digraph* **by** *simp*
next
have (*arcs-ends gen-graph*)$^+$ = {}
using *trancl-empty gen-graph-empty-arcs* **by** (*simp add*: *arcs-ends-def*)
then show *DAG-axioms gen-graph*

by (*simp add*: *DAG-axioms.intro*)
next
fix *u v e*
have *wf-digraph.arc gen-graph e* $(u, v) \equiv$ *False*
using *wf-digraph.arc-def gen-graph-empty-arcs*
by (*simp add*: *wf-digraph.arc-def wf-digraph-def*)
then show *wf-digraph.arc gen-graph e* $(u, v) \Longrightarrow$
$u \rightarrow^{+}{}_{\textit{pre-digraph.del-arc gen-graph e}}\ v \Longrightarrow$ *False*
by *simp*
next
have *refl*: *genesis-node* $\rightarrow^{*}{}_{\textit{gen-graph}}$ *genesis-node*
using *gen-gen rtrancl-on-refl*
by (*simp add*: *reachable-def*)
have $\forall r.\ r \in$ *verts gen-graph* $\longrightarrow r \rightarrow^{*}{}_{\textit{gen-graph}}$ *genesis-node*
proof *safe*
fix *r*
assume $r \in$ *verts gen-graph*
then have $r =$ *genesis-node*
using *gen-gen* **by** *auto*
then show $r \rightarrow^{*}{}_{\textit{gen-graph}}$ *genesis-node*
by (*simp add*: *local.refl*)
qed
then show $\exists p \in$ *verts gen-graph*.
$(\forall r.\ r \in$ *verts gen-graph* $\longrightarrow r \rightarrow^{+}{}_{\textit{gen-graph}}\ p \lor r = p)$
by (*simp add*: *gen-gen*)
qed

lemma (**in** *blockDAG*) *no-empty-blockDAG*:
shows *card* (*verts G*) > 0
proof –
have $\exists p.\ p \in$ *verts G*
using *genesis-in-verts* **by** *auto*
then show *card* (*verts G*) > 0
using *card-gt-0-iff finite-verts* **by** *blast*
qed

lemma (**in** *blockDAG*) *gen-graph-all-one*:
card (*verts* (*G*)) $= 1 \longleftrightarrow G =$ *gen-graph*
using *card-1-singletonE gen-graph-def genesis-in-verts*
induce-eq-iff-induced induced-subgraph-refl singletonD gen-graph-def genesis-node-def
by (*metis gen-gen genesis-existAlt is-genesis-node.simps less-one linorder-neqE-nat neq0-conv no-empty-blockDAG tips-unequal-gen-exist*)

lemma *blockDAG-nat-induct*[*consumes 1*, *case-names base step*]:
assumes
bD: *blockDAG Z*
and
cases: $\bigwedge V.$ (*blockDAG V* $\Longrightarrow$ *card* (*verts V*) $= 1 \Longrightarrow P\ V$)
$\bigwedge W\ c.$ ($\bigwedge V.$ (*blockDAG V* $\Longrightarrow$ *card* (*verts V*) $= c \Longrightarrow P\ V$))

$\Longrightarrow$ (*blockDAG W* $\Longrightarrow$ *card* (*verts W*) = *Suc c* $\Longrightarrow$ *P W*)
shows *P Z*
proof –
have *bG*: *card* (*verts Z*) > *0* **using** *bD blockDAG.no-empty-blockDAG* **by** *auto*
show *?thesis*
using *bG bD*
proof (*induction card* (*verts Z*) *arbitrary*: *Z rule*: *Nat.nat-induct-non-zero*)
case *1*
then show *?case* **using** *cases*(*1*) **by** *auto*
next
case *su*: (*Suc n*)
show *?case*
by (*metis local.cases*(*2*) *su.hyps*(*2*) *su.hyps*(*3*) *su.prems*)
qed
qed

lemma *blockDAG-nat-less-induct*[*consumes 1*, *case-names base step*]:
assumes
bD: *blockDAG Z*
and
cases: $\bigwedge$*V*. (*blockDAG V* $\Longrightarrow$ *card* (*verts V*) = *1* $\Longrightarrow$ *P V*)
$\bigwedge$*W c*. ($\bigwedge$*V*. (*blockDAG V* $\Longrightarrow$ *card* (*verts V*) < *c* $\Longrightarrow$ *P V*))
$\Longrightarrow$ (*blockDAG W* $\Longrightarrow$ *card* (*verts W*) = *c* $\Longrightarrow$ *P W*)
shows *P Z*
proof –
have *bG*: *card* (*verts Z*) > *0* **using** *blockDAG.no-empty-blockDAG assms*(*1*) **by** *auto*
show *P Z*
using *bD bG*
proof (*induction card* (*verts Z*) *arbitrary*: *Z rule*: *less-induct*)
fix *Z*::(*'a*, *'b*) *pre-digraph*
assume *a*:
($\bigwedge$*Za*. *card* (*verts Za*) < *card* (*verts Z*) $\Longrightarrow$ *blockDAG Za* $\Longrightarrow$ *0* < *card* (*verts Za*) $\Longrightarrow$ *P Za*)
assume *blockDAG Z*
then show *P Z* **using** *a cases*
by (*metis blockDAG.no-empty-blockDAG*)
qed
qed

lemma (**in** *blockDAG*) *blockDAG-size-cases*:
obtains (*one*) *card* (*verts G*) = *1*
| (*more*) *card* (*verts G*) > *1*
using *no-empty-blockDAG*
by *linarith*

lemma (**in** *blockDAG*) *blockDAG-cases-one*:
shows *card* (*verts G*) = *1* $\longrightarrow$ (*G* = *gen-graph*)

proof (*safe*)
assume *one*: *card* (*verts* *G*) = *1*
then have *blockDAG.genesis-node* *G* ∈ *verts* *G*
by (*simp add*: *genesis-in-verts*)
then have *only*: *verts* *G* = {*blockDAG.genesis-node* *G*}
by (*metis one card-1-singletonE insert-absorb singleton-insert-inj-eq′*)
then have *verts-equal*: *verts* *G* = *verts* (*blockDAG.gen-graph* *G*)
using *blockDAG-axioms one blockDAG.gen-graph-def induce-subgraph-def induced-induce blockDAG.genesis-in-verts*
by (*simp add*: *blockDAG.gen-graph-def*)
have *arcs* *G* ={}
proof (*rule ccontr*)
assume *not-empty*: *arcs* *G* ≠{}
then obtain *z* **where** *part-of*: *z* ∈ *arcs* *G*
by *auto*
then have *tail*: *tail* *G* *z* ∈ *verts* *G*
using *wf-digraph-def blockDAG-def DAG-def digraph-def blockDAG-axioms nomulti-digraph.axioms*(*1*)
by *metis*
also have *head*: *head* *G* *z* ∈ *verts* *G*
by (*metis* (*no-types*) *DAG-def blockDAG-axioms blockDAG-def digraph-def nomulti-digraph.axioms*(*1*) *part-of wf-digraph-def*)
then have *tail* *G* *z* = *head* *G* *z*
using *tail only* **by** *simp*
then have ¬ *loopfree-digraph-axioms* *G*
unfolding *loopfree-digraph-axioms-def*
using *part-of only DAG-def digraph-def*
by *auto*
then show *False*
using *DAG-def digraph-def blockDAG-axioms blockDAG-def loopfree-digraph-def* **by** *metis*
qed
then have *arcs* *G* = *arcs* (*blockDAG.gen-graph* *G*)
by (*simp add*: *blockDAG-axioms blockDAG.gen-graph-empty-arcs*)
then show *G* = *gen-graph*
unfolding *blockDAG.gen-graph-def*
using *verts-equal blockDAG-axioms induce-subgraph-def blockDAG.gen-graph-def* **by** *fastforce*
qed

lemma (**in** *blockDAG*) *blockDAG-cases-more*:
shows *card* (*verts* *G*) > *1* ⟷ (∃ *b* *H*. (*blockDAG* *H* ∧ *b* ∈ *verts* *G* ∧ *del-vert* *b* = *H*))
proof *safe*
assume *card* (*verts* *G*) > *1*
then have *b1*: *1* < *card* (*verts* *G*) **using** *no-empty-blockDAG* **by** *linarith*
obtain *x* **where** *x-in*: *x* ∈ (*verts* *G*) ∧ *is-genesis-node* *x*
using *genesis genesisAlt genesis-node-def* **by** *blast*
then have *0* < *card* ((*verts* *G*) − {*x*}) **using** *card-Suc-Diff1 x-in finite-verts b1*

by *auto*
  then have $((verts\ G) - \{x\}) \neq \{\}$ **using** *card-gt-0-iff* **by** *blast*
  then obtain y **where** *y-def*:$y \in (verts\ G) - \{x\}$ **by** *auto*
  then have *uneq*: $y \neq x$ **by** *auto*
  have *y-in*: $y \in (verts\ G)$ **using** *y-def* **by** *simp*
  then have *reachable1 G y x* **using** *is-genesis-node.simps x-in*
    reachable-neq-reachable1 uneq **by** *simp*
  then have ¬ *is-tip G x*
    using *y-in* **by** *force*
  then obtain z **where** *z-def*: $z \in (verts\ G) - \{x\} \wedge$ *is-tip G z* **using** *tips-exist*
  is-tip.simps **by** *auto*
  then have *uneq*: $z \neq x$ **by** *auto*
  have *z-in*: $z \in verts\ G$ **using** *z-def* **by** *simp*
  have ¬ *is-genesis-node z*
  proof (*rule ccontr*, *safe*)
    assume *is-genesis-node z*
    then have $x = z$ **using** *unique-genesis x-in* **by** *auto*
    then show *False* **using** *uneq* **by** *simp*
  qed
  then have *blockDAG* (*del-vert z*) **using** *del-tips-bDAG z-def* **by** *simp*
  then show ($\exists\, b\ H.$ *blockDAG H* $\wedge\ b \in verts\ G \wedge$ *del-vert* $b = H$) **using** *z-def*
by *auto*
next
  fix b **and** H::('a,'b) *pre-digraph*
  assume *bD*: *blockDAG* (*del-vert b*)
  assume *b-in*: $b \in verts\ G$
  show *card* (*verts G*) $>$ *1*
  proof (*rule ccontr*)
    assume ¬ *1* $<$ *card* (*verts G*)
    then have *1* $=$ *card* (*verts G*) **using** *no-empty-blockDAG* **by** *linarith*
  then have *card* (*verts* (*del-vert b*)) $=$ *0* **using** *b-in del-vert-def* **by** *auto*
  then have ¬ *blockDAG* (*del-vert b*) **using** *bD blockDAG.no-empty-blockDAG*
    by (*metis less-nat-zero-code*)
  then show *False* **using** *bD* **by** *simp*
  qed
qed

lemma (**in** *blockDAG*) *blockDAG-cases*:
  obtains (*base*) (G = *gen-graph*)
  | (*more*) ($\exists\, b\ H.$ (*blockDAG H* $\wedge\ b \in verts\ G \wedge$ *del-vert* $b = H$))
  using *blockDAG-cases-one blockDAG-cases-more*
    blockDAG-size-cases **by** *auto*

lemma *blockDAG-induct*[*consumes 1*, *case-names fund base step*]:
  assumes *base*: *blockDAG G*
  assumes *cases*: $\bigwedge V$::('a,'b) *pre-digraph*. *blockDAG V* $\Longrightarrow P$ (*blockDAG.gen-graph*
V)
    $\bigwedge H$::('a,'b) *pre-digraph*.
  ($\bigwedge b$::'a. *blockDAG* (*pre-digraph.del-vert H b*) $\Longrightarrow b \in verts\ H \Longrightarrow P$(*pre-digraph.del-vert*

```
H b))
  ⟹ (blockDAG H ⟹ P H)
    shows P G
proof(induct-tac G rule:blockDAG-nat-induct)
  show blockDAG G using assms(1) by simp
next
  fix V::('a,'b) pre-digraph
  assume bD: blockDAG V
  and card (verts V) = 1
  then have V = blockDAG.gen-graph V
    using blockDAG.blockDAG-cases-one equal-refl  by auto
  then show P V using bD cases(1)
    by metis
next
  fix c and W::('a,'b) pre-digraph
  show (⋀V. blockDAG V ⟹ card (verts V) = c ⟹ P V) ⟹
         blockDAG W ⟹ card (verts W) = Suc c ⟹ P W
  proof -
    assume ind: ⋀V. (blockDAG V ⟹ card (verts V) = c ⟹ P V)
    and bD: blockDAG W
    and  size: card (verts W) = Suc c
    have  assm2: ⋀b. blockDAG (pre-digraph.del-vert W b)
          ⟹ b ∈ verts W ⟹ P(pre-digraph.del-vert W b)
    proof -
      fix b
      assume bD2: blockDAG (pre-digraph.del-vert W b)
      assume in-verts: b ∈ verts W
      have verts (pre-digraph.del-vert W b) = verts W − {b}
        by (simp add: pre-digraph.verts-del-vert)
      then have card ( verts (pre-digraph.del-vert W b)) = c
       using in-verts fin-digraph.finite-verts bD subs fin-digraph.fin-digraph-del-vert

         size
        by (simp add: fin-digraph.finite-verts subs
            DAG.axioms assms(1) digraph.axioms)
      then show P (pre-digraph.del-vert W b) using ind bD2 by auto
    qed
    show ?thesis using cases(2)
      by (metis assm2 bD)
  qed
qed

function genesis-nodeAlt:: ('a::linorder,'b) pre-digraph ⇒ 'a
  where genesis-nodeAlt G = (if (¬ blockDAG G) then undefined else
  if (card (verts G ) = 1) then (hd (sorted-list-of-set (verts G)))
  else genesis-nodeAlt (reduce-past G ((hd (sorted-list-of-set (tips G))))))
  by auto
termination proof
  let ?R = measure ( λG. (card (verts G)))
```

show *wf ?R* **by** *auto*
next
fix *G* ::(*'a*::*linorder*,*'b*) *pre-digraph*
assume ¬ ¬ *blockDAG G*
then have *bD*: *blockDAG G* **by** *simp*
assume *card* (*verts G*) ≠ *1*
then have *bG*: *card* (*verts G*) > *1* **using** *bD blockDAG.blockDAG-size-cases* **by** *auto*
have *set* (*sorted-list-of-set* (*tips G*)) = *tips G*
by (*simp add*: *bD subs tips-def fin-digraph.finite-verts*)
then have *hd* (*sorted-list-of-set* (*tips G*)) ∈ *tips G*
using *hd-in-set bD tips-def bG blockDAG.tips-unequal-gen-exist*
empty-iff empty-set mem-Collect-eq
by (*metis* (*mono-tags*, *lifting*))
then show (*reduce-past G* (*hd* (*sorted-list-of-set* (*tips G*))), *G*) ∈ *measure* (λ*G*. *card* (*verts G*))
using *blockDAG.reduce-less bD*
using *tips-def* **by** *fastforce*
qed

lemma *genesis-nodeAlt-one-sound*:
assumes *bD*: *blockDAG G*
and *one*: *card* (*verts G*) = *1*
shows *blockDAG.is-genesis-node G* (*genesis-nodeAlt G*)
proof −
have *exone*: ∃! *x*. *x* ∈ (*verts G*)
using *bD one blockDAG.genesis-in-verts blockDAG.genesis-unique-exists blockDAG.reduce-less*
blockDAG.reduce-past-dagbased less-nat-zero-code less-one **by** *metis*
then have *sorted-list-of-set* (*verts G*) ≠ []
by (*metis card.infinite card-0-eq finite.emptyI one*
sorted-list-of-set-empty sorted-list-of-set-inject zero-neq-one)
then have *genesis-nodeAlt G* ∈ *verts G* **using** *hd-in-set genesis-nodeAlt.simps bD exone*
by (*metis one set-sorted-list-of-set sorted-list-of-set.infinite*)
then show *one-sound*: *blockDAG.is-genesis-node G* (*genesis-nodeAlt G*)
using *bD one*
by (*metis blockDAG.blockDAG-size-cases blockDAG.reduce-less*
blockDAG.reduce-past-dagbased less-one not-one-less-zero)
qed

lemma *genesis-nodeAlt-sound* :
assumes *blockDAG G*
shows *blockDAG.is-genesis-node G* (*genesis-nodeAlt G*)
proof(*induct-tac G rule*:*blockDAG-nat-less-induct*)
show *blockDAG G* **using** *assms* **by** *simp*
next
fix *V*::(*'a*,*'b*) *pre-digraph*
assume *bD*: *blockDAG V*
assume *one*: *card* (*verts V*) = *1*

then show *blockDAG.is-genesis-node V* (*genesis-nodeAlt V*)
using *genesis-nodeAlt-one-sound bD*
by *blast*
next
fix *W*::(*'a*,*'b*) *pre-digraph*
fix *c*::*nat*
assume *basis*:
($\bigwedge$ *V*::(*'a*,*'b*) *pre-digraph*. *blockDAG V* $\Longrightarrow$ *card* (*verts V*) < *c* $\Longrightarrow$
blockDAG.is-genesis-node V (*genesis-nodeAlt V*))
assume *bD*: *blockDAG W*
assume *cd*: *card* (*verts W*) = *c*
consider (*one*) *card* (*verts W*) = *1* | (*more*) *card* (*verts W*) > *1*
using *bD blockDAG.blockDAG-size-cases* **by** *blast*
then show *blockDAG.is-genesis-node W* (*genesis-nodeAlt W*)
proof(*cases*)
case *one*
then show *?thesis* **using** *genesis-nodeAlt-one-sound bD*
by *blast*
next
case *more*
then have *not-one*: *1* ≠ *card* (*verts W*) **by** *auto*
have *se*: *set* (*sorted-list-of-set* (*tips W*)) = *tips W*
by (*simp add*: *bD subs tips-def fin-digraph.finite-verts*)
obtain *a* **where** *a-def*: *a* = *hd* (*sorted-list-of-set* (*tips W*))
by *simp*
have *tip*: *a* ∈ *tips W*
using *se a-def hd-in-set bD tips-def more blockDAG.tips-unequal-gen-exist*
empty-iff empty-set mem-Collect-eq
by (*metis* (*mono-tags*, *lifting*))
then have *ver*: *a* ∈ *verts W*
by (*simp add*: *tips-def a-def*)
then have *card* (*verts* (*reduce-past W a*)) < *card* (*verts W*)
using *more cd blockDAG.reduce-less bD*
by *metis*
then have *cd2*: *card* (*verts* (*reduce-past W a*)) < *c*
using *cd* **by** *simp*
have *n-gen*: ¬ *blockDAG.is-genesis-node W a*
using *blockDAG.tips-unequal-gen bD more tip tips-def Collect-mem-eq* **by** *fastforce*
then have *bD2*: *blockDAG* (*reduce-past W a*)
using *blockDAG.reduce-past-dagbased ver bD* **by** *auto*
have *ff*: *blockDAG.is-genesis-node* (*reduce-past W a*)
(*genesis-nodeAlt* (*reduce-past W a*)) **using** *cd2 basis bD2 more*
by *blast*
have *rec*: *genesis-nodeAlt W* = *genesis-nodeAlt* (*reduce-past W* (*hd* (*sorted-list-of-set* (*tips W*))))
using *genesis-nodeAlt.simps not-one bD*
by *metis*
show *?thesis* **using** *rec ff bD n-gen ver blockDAG.reduce-past-gen-eq a-def* **by**

metis
qed
qed

end

theory *Spectre*
imports *Main Graph-Theory.Graph-Theory blockDAG*
begin

Based on the SPECTRE paper by Sompolinsky, Lewenberg and Zohar 2016

4 Spectre

4.1 Definitions

Function to check and break occuring ties

```
fun tie-break-int:: 'a::linorder ⇒ 'a ⇒ int ⇒ int
  where tie-break-int a b i =
 (if i=0 then (if (b < a) then −1 else 1) else
             (if i > 0 then 1 else −1))
```

Function to check if all entries of a list are zero

```
fun zero-list:: int list ⇒ bool
  where zero-list [] = True
  | zero-list (x # xs) = ((x = 0) ∧ zero-list xs)
```

Function given a list of votes, sums them up if not only zeros, otherwise no_vote

```
fun sumlist-break :: 'a::linorder ⇒'a ⇒ int list ⇒ int
  where sumlist-break a b L = (if (zero-list L) then 0 else
   tie-break-int a b (sum-list L))
```

Spectre core algorithm, $vote_SpectreVabc$ returns 1 if a votes in favour of b (or $b = c$), -1 if a votes in favour of c, 0 otherwise

```
function vote-Spectre :: ('a::linorder,'b) pre-digraph ⇒'a ⇒ 'a ⇒ 'a ⇒ int
  where
  vote-Spectre V a b c = (
  if (¬ blockDAG V ∨ a ∉ verts V ∨ b ∉ verts V ∨ c ∉ verts V) then 0 else
  if (b=c)  then 1 else
  if (((a →+ V b) ∨ a = b) ∧ ¬(a →+ V c)) then 1  else
  if (((a →+ V c) ∨ a = c) ∧ ¬(a →+ V b)) then −1 else
  if ((a →+ V b) ∧ (a →+ V c)) then
   (sumlist-break b c (map (λi.
 (vote-Spectre (reduce-past V a) i b c)) (sorted-list-of-set (past-nodes V a))))
```

else
sumlist-break b c (*map* ($\lambda i.$
(*vote-Spectre V i b c*)) (*sorted-list-of-set* (*future-nodes V a*))))
by *auto*
termination
proof
let *?R* = *measures* [($\lambda(V, a, b, c)$. (*card* (*verts V*))), ($\lambda(V, a, b, c)$. *card* $\{e.\ e \rightarrow^*{}_V\ a\}$)]
show *wf ?R*
by *simp*
next
fix V::($'a$::*linorder*, $'b$) *pre-digraph*
fix *x a b c*
assume *bD*: $\neg$ ($\neg$ *blockDAG V* $\vee$ $a \notin$ *verts V* $\vee$ $b \notin$ *verts V* $\vee$ $c \notin$ *verts V*)
then have $a \in$ *verts V* **by** *simp*
then have *card* (*verts* (*reduce-past V a*)) < *card* (*verts V*)
using *bD blockDAG.reduce-less*
by *metis*
then show ((*reduce-past V a*, *x*, *b*, *c*), *V*, *a*, *b*, *c*)
$\in$ *measures*
[$\lambda(V, a, b, c)$. *card* (*verts V*),
$\lambda(V, a, b, c)$. *card* $\{e.\ e \rightarrow^*{}_V\ a\}$]
by *simp*
next
fix V::($'a$::*linorder*, $'b$) *pre-digraph*
fix *x a b c*
assume *bD*: $\neg$ ($\neg$ *blockDAG V* $\vee$ $a \notin$ *verts V* $\vee$ $b \notin$ *verts V* $\vee$ $c \notin$ *verts V*)
then have *a-in*: $a \in$ *verts V* **using** *bD* **by** *simp*
assume $x \in$ *set* (*sorted-list-of-set* (*future-nodes V a*))
then have $x \in$ *future-nodes V a* **using** *DAG.finite-future*
set-sorted-list-of-set bD subs
by *metis*
then have *rr*: $x \rightarrow^+{}_V\ a$ **using** *future-nodes.simps bD mem-Collect-eq*
by *simp*
then have *a-not*: $\neg\ a \rightarrow^*{}_V\ x$ **using** *bD DAG.unidirectional subs* **by** *metis*
have *bD2*: *blockDAG V* **using** *bD* **by** *simp*
have $\forall x.\ \{e.\ e \rightarrow^*{}_V\ x\} \subseteq$ *verts V* **using** *subs bD2 subsetI*
wf-digraph.reachable-in-verts(*1*) *mem-Collect-eq*
by *metis*
then have *fin*: $\forall x.$ *finite* $\{e.\ e \rightarrow^*{}_V\ x\}$ **using** *subs bD2 fin-digraph.finite-verts*
finite-subset
by *metis*
have $x \rightarrow^*{}_V\ a$ **using** *rr wf-digraph.reachable1-reachable subs bD2* **by** *metis*
then have $\{e.\ e \rightarrow^*{}_V\ x\} \subseteq \{e.\ e \rightarrow^*{}_V\ a\}$ **using** *rr*
wf-digraph.reachable-trans Collect-mono subs bD2 **by** *metis*
then have $\{e.\ e \rightarrow^*{}_V\ x\} \subset \{e.\ e \rightarrow^*{}_V\ a\}$ **using** *a-not*
subs bD2 a-in mem-Collect-eq psubsetI wf-digraph.reachable-refl
by *metis*
then have *card* $\{e.\ e \rightarrow^*{}_V\ x\}$ < *card* $\{e.\ e \rightarrow^*{}_V\ a\}$ **using** *fin*

by (*simp add*: *psubset-card-mono*)
then show $((V, x, b, c), V, a, b, c)$
$\in$ *measures*
$[\lambda(V, a, b, c).\ card\ (verts\ V), \lambda(V, a, b, c).\ card\ \{e.\ e \rightarrow^*{}_V\ a\}]$
by *simp*
qed

Given $vote_Spectre$ calculate if $a < b$ for arbitrary nodes

definition *Spectre-Order* :: ('*a*::*linorder*,'*b*) *pre-digraph* $\Rightarrow$ '*a* $\Rightarrow$ '*a* $\Rightarrow$ *bool*
where *Spectre-Order G a b* = (*sumlist-break a b* (*map* (λ*i*.
(*vote-Spectre G i a b*)) (*sorted-list-of-set* (*verts G*))) = *1*)

Given $Spectre_Order$ calculate the corresponding relation over the nodes of G

definition *Spectre-Order-Relation* :: ('*a*::*linorder*,'*b*) *pre-digraph* $\Rightarrow$ ('*a* $\times$ '*a*) *set*
where *Spectre-Order-Relation G* $\equiv$ $\{(a,b) \in (verts\ G \times verts\ G).\ Spectre\text{-}Order\ G\ a\ b\}$

4.2 Lemmas

lemma *zero-list-sound*:
zero-list L $\equiv \forall\ a \in set\ L.\ a = 0$
proof(*induct L*, *auto*) **qed**

lemma *sumlist-one-mono*:
assumes $\forall\ x \in set\ L.\ x \geq 0$
and $\exists\ x \in set\ L.\ x > 0$
and $L \neq []$
shows *sumlist-break a b L* = *1*
using *assms*
proof(*induct L*, *simp*)
case (*Cons a2 L*)
then have *nz*: $\neg$ *zero-list* (*a2* # *L*) **using** *assms*
by (*metis less-int-code*(*1*) *zero-list-sound*)
consider (*bg*) *a2* > *0* | *a2* = *0* **using** *Cons*
by (*metis le-less list.set-intros*(*1*))
then show *?case*
proof(*cases*)
case *bg*
then have *sum-list L* $\geq$ *0* **using** *Cons*
by (*simp add*: *sum-list-nonneg*)
then have *sum-list* (*a2* # *L*) > *0* **using** *bg sum-list-def*
by *auto*
then show *?thesis* **using** *nz sumlist-break.simps tie-break-int.simps*
by *auto*
next
case *2*
then have *be*: $\exists\ a \in set\ L.\ 0 < a$ **using** *Cons*
by (*metis less-int-code*(*1*) *set-ConsD*)

then have $L \neq []$ **by** *auto*
then have *sumlist-break a b L* = *1* **using** *Cons be*
by *auto*
then show *?thesis* **using** *sum-list-def 2 sumlist-break.simps nz*
by *auto*
qed
qed

lemma *domain-tie-break*:
shows *tie-break-int a b c* $\in \{-1, 1\}$
using *tie-break-int.simps* **by** *simp*

lemma *domain-sumlist*:
shows *sumlist-break a b c* $\in \{-1, 0, 1\}$
using *insertCI sumlist-break.elims domain-tie-break*
by (*metis insert-commute*)

lemma *domain-sumlist-not-empty*:
assumes $\neg$ *zero-list l*
shows *sumlist-break a b l* $\in \{-1, 1\}$
using *sumlist-break.elims domain-tie-break assms*
by *metis*

lemma *Spectre-casesAlt*:
fixes V:: ($'a$::*linorder*,$'b$) *pre-digraph*
and a :: $'a$::*linorder* **and** b :: $'a$::*linorder* **and** c :: $'a$::*linorder*
obtains (*no-bD*) ($\neg$ *blockDAG* $V \lor a \notin$ *verts* $V \lor b \notin$ *verts* $V \lor c \notin$ *verts* V)
| (*equal*) (*blockDAG* $V \land a \in$ *verts* $V \land b \in$ *verts* $V \land c \in$ *verts* V) $\land b = c$
| (*one*) (*blockDAG* $V \land a \in$ *verts* $V \land b \in$ *verts* $V \land c \in$ *verts* V) $\land$
$b \neq c \land (((a \rightarrow^+{}_V b) \lor a = b) \land \neg(a \rightarrow^+{}_V c))$
| (*two*) (*blockDAG* $V \land a \in$ *verts* $V \land b \in$ *verts* $V \land c \in$ *verts* V) $\land b \neq c$
$\land \neg(((a \rightarrow^+{}_V b) \lor a = b) \land \neg(a \rightarrow^+{}_V c)) \land$
$((a \rightarrow^+{}_V c) \lor a = c) \land \neg(a \rightarrow^+{}_V b)$
| (*three*) (*blockDAG* $V \land a \in$ *verts* $V \land b \in$ *verts* $V \land c \in$ *verts* V) $\land b \neq c$
$\land \neg(((a \rightarrow^+{}_V b) \lor a = b) \land \neg(a \rightarrow^+{}_V c)) \land$
$\neg(((a \rightarrow^+{}_V c) \lor a = c) \land \neg(a \rightarrow^+{}_V b)) \land$
$((a \rightarrow^+{}_V b) \land (a \rightarrow^+{}_V c))$
| (*four*) (*blockDAG* $V \land a \in$ *verts* $V \land b \in$ *verts* $V \land c \in$ *verts* V) $\land b \neq c \land$
$\neg(((a \rightarrow^+{}_V b) \lor a = b) \land \neg(a \rightarrow^+{}_V c)) \land$
$\neg(((a \rightarrow^+{}_V c) \lor a = c) \land \neg(a \rightarrow^+{}_V b)) \land$
$\neg((a \rightarrow^+{}_V b) \land (a \rightarrow^+{}_V c))$
by *auto*

lemma *Spectre-theo*:
assumes *P 0*

and P *1*
and P (-1)
and P (*sumlist-break b c* (*map* (λ*i*.
(*vote-Spectre* (*reduce-past V a*) *i b c*)) (*sorted-list-of-set* ((*past-nodes V a*)))))
and P (*sumlist-break b c* (*map* (λ*i*.
(*vote-Spectre V i b c*)) (*sorted-list-of-set* (*future-nodes V a*))))
shows P (*vote-Spectre V a b c*)
using *assms vote-Spectre.simps*
by (*metis* (*mono-tags*, *lifting*))

lemma *domain-Spectre*:
shows *vote-Spectre V a b c* $\in \{-1, 0, 1\}$
proof(*rule Spectre-theo*, *simp*, *simp*, *simp*, *metis domain-sumlist*, *metis domain-sumlist*)
qed

lemma *antisymmetric-tie-break*:
shows $b \neq c \implies$ *tie-break-int b c i* = − *tie-break-int c b* $(-i)$
unfolding *tie-break-int.simps* **using** *less-not-sym* **by** *auto*

lemma *antisymmetric-sumlist*:
shows $b \neq c \implies$ *sumlist-break b c l* = − *sumlist-break c b* (*map* ($\lambda x.\ -x$) *l*)
proof(*induct l*, *simp*)
case (*Cons a l*)
have *sum-list* (*map uminus* (*a* # *l*)) = − *sum-list* (*a* # *l*)
by (*metis map-ident map-map uminus-sum-list-map*)
moreover have *zero-list* (*map* ($\lambda x.\ -x$) *l*) $\equiv$ *zero-list l*
proof(*induct l*, *auto*) **qed**
ultimately show *?case* **using** *sumlist-break.simps antisymmetric-tie-break Cons*
by *auto*
qed

lemma *vote-Spectre-antisymmetric*:
shows $b \neq c \implies$ *vote-Spectre V a b c* = − (*vote-Spectre V a c b*)
proof(*induction V a b c rule*: *vote-Spectre.induct*)
case (*1 V a b c*)
show *vote-Spectre V a b c* = − *vote-Spectre V a c b*
proof(*cases a b c V rule*:*Spectre-casesAlt*)
case *no-bD*
then show *?thesis* **by** *fastforce*
next
case *equal*
then show *?thesis* **using** *1* **by** *simp*
next
case *one*

then show *?thesis* **by** *auto*
next
case *two*
then show *?thesis* **by** *fastforce*
next
case *three*
then have *ff*: *vote-Spectre V a b c* = (*sumlist-break b c* (*map* (*λi.*
(*vote-Spectre* (*reduce-past V a*) *i b c*)) (*sorted-list-of-set* (*past-nodes V a*))))
by (*metis* (*mono-tags*, *lifting*) *vote-Spectre.elims*)
have *ff2*: *vote-Spectre V a c b* = (*sumlist-break c b* (*map* (*λi.*
(− *vote-Spectre* (*reduce-past V a*) *i b c*)) (*sorted-list-of-set* (*past-nodes V a*))))
using *three 1 vote-Spectre.simps map-eq-conv*
by (*smt* (*verit*, *ccfv-SIG*))
have (*map* (*λi.* − *vote-Spectre* (*reduce-past V a*) *i b c*) (*sorted-list-of-set* (*past-nodes V a*)))
= (*map uminus* (*map* (*λi. vote-Spectre* (*reduce-past V a*) *i b c*)
(*sorted-list-of-set* (*past-nodes V a*))))
using *map-map* **by** *auto*
then have *vote-Spectre V a c b* = − (*sumlist-break b c* (*map* (*λi.*
(*vote-Spectre* (*reduce-past V a*) *i b c*)) (*sorted-list-of-set* (*past-nodes V a*))))
using *antisymmetric-sumlist 1 ff2*
by (*metis verit-minus-simplify*(*4*))
then show *?thesis* **using** *ff*
by *presburger*
next
case *four*
then have *ff*: *vote-Spectre V a b c* = *sumlist-break b c* (*map* (*λi.*
(*vote-Spectre V i b c*)) (*sorted-list-of-set* (*future-nodes V a*)))
using *vote-Spectre.simps*
by (*metis* (*mono-tags*, *lifting*))
have *ff2*: *vote-Spectre V a c b* = (*sumlist-break c b* (*map* (*λi.*
(− *vote-Spectre V i b c*)) (*sorted-list-of-set* (*future-nodes V a*))))
using *four 1 vote-Spectre.simps map-eq-conv*
by (*smt* (*z3*))
have (*map* (*λi.* − *vote-Spectre V i b c*) (*sorted-list-of-set* (*future-nodes V a*)))
= (*map uminus* (*map* (*λi. vote-Spectre V i b c*) (*sorted-list-of-set* (*future-nodes V a*))))
using *map-map* **by** *auto*
then have *vote-Spectre V a c b* = − (*sumlist-break b c* (*map* (*λi.*
(*vote-Spectre V i b c*)) (*sorted-list-of-set* (*future-nodes V a*))))
using *antisymmetric-sumlist 1 ff2*
by (*metis verit-minus-simplify*(*4*))
then show *?thesis* **using** *ff*
by *linarith*
qed
qed

lemma *vote-Spectre-reflexive*:

assumes *blockDAG V*
and $a \in$ *verts V*
shows $\forall\, b \in$ *verts V*. *vote-Spectre V b a a* = *1* **using** *vote-Spectre.simps assms*
by *auto*

lemma *Spectre-Order-reflexive*:
assumes *blockDAG V*
and $a \in$ *verts V*
shows *Spectre-Order V a a*
unfolding *Spectre-Order-def*
proof −
obtain *l* **where** *l-def*: *l* = (*map* (λi. *vote-Spectre V i a a*) (*sorted-list-of-set* (*verts V*)))
by *auto*
have *only-one*: *l* = (*map* (λi.*1*) (*sorted-list-of-set* (*verts V*)))
using *l-def vote-Spectre-reflexive assms sorted-list-of-set*(*1*)
by (*simp add*: *fin-digraph.finite-verts subs*)
have *ne*: $l \neq []$
using *blockDAG.no-empty-blockDAG length-map*
by (*metis assms*(*1*) *length-sorted-list-of-set less-numeral-extra*(*3*) *list.size*(*3*) *l-def*)
then have *snn*: ¬ *zero-list l* **using** *only-one*
using *zero-list.elims*(*2*) **by** *fastforce*
have *sum-list l* = *card* (*verts V*) **using** *ne only-one sum-list-map-eq-sum-count*
by (*simp add*: *sum-list-triv*)
then have *sum-list l* > *0* **using** *blockDAG.no-empty-blockDAG assms*(*1*) **by** *simp*
then show *sumlist-break a a* (*map* (λi. *vote-Spectre V i a a*) (*sorted-list-of-set* (*verts V*))) = *1*
using *l-def ne sumlist-break.simps tie-break-int.simps*
list.exhaust verit-comp-simplify1(*1*) *snn* **by** *auto*
qed

lemma *vote-Spectre-one-exists*:
assumes *blockDAG V*
and $a \in$ *verts V*
and $b \in$ *verts V*
shows $\exists\ i \in$ *verts V*. *vote-Spectre V i a b* $\neq$ *0*
proof
show $a \in$ *verts V* **using** *assms*(*2*) **by** *simp*
show *vote-Spectre V a a b* $\neq$ *0*
using *assms*
proof(*cases a b a V rule*: *Spectre-casesAlt*, *simp*, *simp*, *simp*, *simp*)
case *three*
then show *?thesis*
by (*meson DAG.cycle-free blockDAG.axioms*(*1*))
next
case *four*
then show *?thesis*

    by *blast*
  qed
qed

lemma *Spectre-Order-antisym*:
  assumes *blockDAG V*
  and $a \in$ *verts V*
  and $b \in$ *verts V*
  and $a \neq b$
  shows *Spectre-Order V a b* = (¬ (*Spectre-Order V b a*))
proof −
  obtain *wit* **where** *wit-in*: *vote-Spectre V wit a b* ≠ *0* ∧ *wit* ∈ *verts V*
    using *vote-Spectre-one-exists assms*
    by *blast*
  obtain *l* **where** *l-def*: *l* = (*map* (λ*i*. *vote-Spectre V i a b*) (*sorted-list-of-set* (*verts V*)))
    by *auto*
  have *wit* ∈ *set* (*sorted-list-of-set* (*verts V*))
    using *wit-in sorted-list-of-set*(*1*)
    fin-digraph.finite-verts subs
    by (*simp add*: *fin-digraph.finite-verts subs assms*(*1*))
  then have *vote-Spectre V wit a b* ∈ *set l* **unfolding** *l-def*
    by (*metis* (*mono-tags*, *lifting*) *image-eqI list.set-map*)
  then have *ne0*: ¬ *zero-list l* **using** *assms l-def zero-list-sound*
    zero-neq-one wit-in
    by *blast*
  then have *dm*: *sumlist-break a b l* ∈ {−*1*,*1*} **using** *domain-sumlist-not-empty* **by** *auto*
  obtain *l2* **where** *l2-def*: *l2* = (*map* (λ*i*. *vote-Spectre V i b a*) (*sorted-list-of-set* (*verts V*)))
      by *auto*
    have *minus*: *l2* = *map uminus l*
      unfolding *l-def l2-def map-map*
      using *vote-Spectre-antisymmetric assms*(*4*)
      by (*metis comp-apply*)
    then have *ne02*: ¬ *zero-list l2* **using** *ne0 zero-list-sound*
      by *fastforce*
    then have *anti*: *sumlist-break a b l* = − *sumlist-break b a l2* **unfolding** *minus*
      using *antisymmetric-sumlist ne0 assms*(*4*) **by** *metis*
    have *dm2*: *sumlist-break b a l2* ∈ {−*1*,*1*} **using** *ne02 domain-sumlist-not-empty* **by** *auto*
    then show *?thesis* **unfolding** *Spectre-Order-def* **using** *anti l-def dm l2-def*
    add.inverse-inverse empty-iff equal-neg-zero insert-iff zero-neq-one
    by (*metis*)
qed

lemma *Spectre-Order-total*:
  assumes *blockDAG V*
  and $a \in$ *verts V* ∧ $b \in$ *verts V*

shows *Spectre-Order V a b* $\vee$ *Spectre-Order V b a*
proof *safe*
  assume *notB*: $\neg$ *Spectre-Order V b a*
  consider (*eq*) $a = b$| (*neq*) $a \neq b$ **by** *auto*
  then show *Spectre-Order V a b*
  proof (*cases*)
  case *eq*
  then show *?thesis* **using** *Spectre-Order-reflexive assms* **by** *metis*
  next
    case *neq*
     then show *?thesis* **using** *Spectre-Order-antisym notB assms*
       by *blast*
   qed
 qed

lemma *Spectre-Order-Relation-total*:
  assumes *blockDAG G*
  shows *total-on* (*verts G*) (*Spectre-Order-Relation G*)
  unfolding *total-on-def Spectre-Order-Relation-def*
  using *Spectre-Order-total assms*
  by *fastforce*

lemma *Spectre-Order-Relation-reflexive*:
  assumes *blockDAG G*
  shows *refl-on* (*verts G*) (*Spectre-Order-Relation G*)
  unfolding *refl-on-def Spectre-Order-Relation-def*
  using *Spectre-Order-reflexive assms* **by** *fastforce*

lemma *Spectre-Order-Relation-antisym*:
  assumes *blockDAG G*
  shows *antisym* (*Spectre-Order-Relation G*)
  unfolding *antisym-def Spectre-Order-Relation-def*
  using *Spectre-Order-antisym assms* **by** *fastforce*

lemma *vote-Spectre-Preserving*:
  assumes $c \rightarrow^{+}{}_{G}\ b$
  shows *vote-Spectre G a b c* $\in \{0,1\}$
  using *assms*
proof(*induction G a b c rule*: *vote-Spectre.induct*)
  case (*1 V a b c*)
  then show *?case*
  proof(*cases a b c V rule*:*Spectre-casesAlt*)
  case *no-bD*
    then show *?thesis* **by** *auto*
  next

```
  case equal
  then show ?thesis  by simp
  next
    case one
    then show ?thesis by auto
  next
    case two
    then show ?thesis
      by (metis local.1.prems trancl-trans)
  next
    case three
    then have b ∈ past-nodes V a by auto
    also have c ∈ past-nodes V a using three by auto
    ultimately have c →+reduce-past V a b using DAG.reduce-past-path2 three 1
      by (metis blockDAG.axioms(1))
    then have all1:∀x. x ∈ set (sorted-list-of-set (past-nodes V a)) ⟶
          vote-Spectre (reduce-past V a) x b c ∈ {0, 1} using 1 three by auto
     obtain the-map where the-map-in:
     the-map = (map (λi. vote-Spectre (reduce-past V a) i b c)
    (sorted-list-of-set (past-nodes V a))) by auto
     consider (zero-l) zero-list  the-map |
              (n-zero-l)  ¬ zero-list  the-map by auto
     then have sumlist-break b c (map (λi. vote-Spectre (reduce-past V a) i b c)
      (sorted-list-of-set (past-nodes V a))) ∈ {0,1}
     proof(cases)
      case zero-l
      then show ?thesis unfolding the-map-in by auto
    next
        case n-zero-l
        then have nem: the-map
          ≠ [] using zero-list-sound
          zero-list.simps(1) the-map-in
          by metis
           have exune: ∃x ∈ set the-map. x  ≠ 0 using n-zero-l zero-list-sound
the-map-in
          by blast
        have all01-1: ∀x ∈ set the-map. x ∈ {0,1}
          unfolding the-map-in set-map
          using all1
          by blast
        then have ∃x ∈ set the-map. x = 1 using exune
          by blast
        then have ∃x ∈ set the-map. x > 0
          using zero-less-one by blast
        moreover have ∀x ∈ set the-map. x ≥ 0 using all01-1
          by (metis empty-iff insert-iff less-int-code(1) not-le-imp-less zero-le-one)
         ultimately show ?thesis using nem unfolding the-map-in using sum-
list-one-mono
          by blast
```

```
      qed
    then show ?thesis using three
      by simp
  next
     case four
     then have all01: ∀ a2. a2 ∈ set (sorted-list-of-set (future-nodes V a)) ⟶
                             vote-Spectre V a2 b c ∈ {0,1}
       using 1
       by metis
     obtain the-map where the-map-in:
      the-map = (map (λi. vote-Spectre V i b c) (sorted-list-of-set (future-nodes V
a))) by auto
     consider (zero-l) zero-list  the-map |
             (n-zero-l)  ¬ zero-list  the-map by auto
     then have sumlist-break b c (map (λi. vote-Spectre V i b c)
      (sorted-list-of-set (future-nodes V a))) ∈ {0,1}
     proof(cases)
      case zero-l
      then show ?thesis unfolding the-map-in by auto
    next
        case n-zero-l
        then have nem: the-map
           ≠ [] using zero-list-sound
          zero-list.simps(1) the-map-in
          by metis
           have exune: ∃ x ∈ set the-map. x  ≠ 0 using n-zero-l zero-list-sound
the-map-in
          by blast
        have all01-2: ∀ x ∈ set the-map. x ∈ {0,1}
          unfolding the-map-in set-map
          using all01
          by blast
        then have ∃ x ∈ set the-map. x = 1 using exune
          by blast
        then have ∃ x ∈ set the-map. x > 0
          using zero-less-one by blast
        moreover have ∀ x ∈ set the-map. x ≥ 0 using all01-2
          by (metis empty-iff insert-iff less-int-code(1) not-le-imp-less zero-le-one)
         ultimately show ?thesis using nem unfolding the-map-in using sum-
list-one-mono
          by blast
      qed
     then show ?thesis using vote-Spectre.simps
       by (simp add: four)
   qed
 qed
```

lemma *Spectre-Order-Preserving*:
  assumes *blockDAG G*
  and $b \rightarrow^{+}{}_{G}\ a$
  shows *Spectre-Order G a b*
proof –
  have *set-ordered*: *set* (*sorted-list-of-set* (*verts G*)) = *verts G*
    using *assms*(*1*) *subs fin-digraph.finite-verts*
    sorted-list-of-set **by** *auto*
  have *a-in*: $a \in$ *verts G* **using** *wf-digraph.reachable1-in-verts*(*2*) *assms subs*
    by *metis*
  have *b-in*: $b \in$ *verts G* **using** *wf-digraph.reachable1-in-verts*(*1*) *assms subs*
    by *metis*
  obtain *the-map* **where** *the-map-in*:
      the-map = (*map* (λi. *vote-Spectre G i a b*) (*sorted-list-of-set* (*verts G*))) **by**
auto
  obtain *wit* **where** *wit-in*: *wit* $\in$ *verts G* **and** *wit-vote*: *vote-Spectre G wit a b* $\neq$
0
    using *vote-Spectre-one-exists a-in b-in assms*(*1*)
    by *blast*
  have (*vote-Spectre G wit a b*) $\in$ *set the-map*
    unfolding *the-map-in set-map*
    using *assms*(*1*) *fin-digraph.finite-verts*
  subs sorted-list-of-set(*1*) *wit-in image-iff*
    by *metis*
  then have *exune*: $\exists\, x \in$ *set the-map*. $x \neq 0$
    using *wit-vote* **by** *blast*
  have *all01*: $\forall\, x \in$ *set the-map*. $x \in \{0,1\}$
   unfolding *set-ordered the-map-in set-map* **using** *vote-Spectre-Preserving assms*(*2*)
image-iff
    by (*metis* (*no-types*, *lifting*))
  then have $\exists\, x \in$ *set the-map*. $x = 1$ **using** *exune*
        by *blast*
  then have $\exists\, x \in$ *set the-map*. $x > 0$
    using *zero-less-one* **by** *blast*
  moreover have $\forall\, x \in$ *set the-map*. $x \geq 0$ **using** *all01*
    by (*metis empty-iff insert-iff less-int-code*(*1*) *not-le-imp-less zero-le-one*)
  ultimately show *?thesis* **unfolding** *the-map-in Spectre-Order-def* **using** *sum-list-one-mono*
      empty-iff set-empty
    by (*metis*)
qed

lemma *Spectre-Order-Relation-Preserving*:
  assumes *blockDAG G*
  and $b \rightarrow^{+}{}_{G}\ a$
shows (*a*,*b*) $\in$ (*Spectre-Order-Relation G*)
  unfolding *Spectre-Order-Relation-def*
  using *assms wf-digraph.reachable1-in-verts subs*

Spectre-Order-Preserving
SigmaI case-prodI mem-Collect-eq **by** *fastforce*
end

theory *Composition*
imports *Main blockDAG*
begin

5 Composition

locale *composition* = *blockDAG* +
fixes C ::'*a set*
assumes $C \subseteq$ *verts* G
and *blockDAG* $(G \upharpoonright C)$
and *same-rel*: $\forall v \in ((verts\ G) - C)$.
$(\forall c \in C.\ (c \rightarrow^+{}_G\ v)) \vee (\forall c \in C.\ (v \rightarrow^+{}_G\ c))$
$\vee\ (\forall c \in C.\ \neg(v \rightarrow^+{}_G\ c) \wedge\ \neg(v \rightarrow^+{}_G\ c))$

locale *compositionGraph* = *blockDAG* +
fixes G' ::('*a set*, '*b*) *pre-digraph*
assumes $\forall C \in (verts\ G')$. *composition* G C
and $\forall\ C1 \in (verts\ G').\ \forall\ C2 \in (verts\ G').\ C1 \cap C2 \neq \{\} \longrightarrow C1 = C2$
and $\bigcup\ (verts\ G') = verts\ G$

5.1 Functions and Definitions

5.2 Lemmas

lemma (**in** *blockDAG*) *trivialComposition*:
assumes C = *verts* G
shows *composition* G C
proof –
show *composition* G C
unfolding *composition-axioms-def composition-def*
proof
show *blockDAG* G **using** *blockDAG-axioms* **by** *simp*
next
have *subset*: $C \subseteq$ *verts* G **using** *assms* **by** *auto*
then have $G \upharpoonright C = G$ **unfolding** *assms induce-subgraph-def*
using *induce-eq-iff-induced induced-subgraph-refl assms* **by** *auto*
then have *bD*: *blockDAG* $(G \upharpoonright C)$ **using** *blockDAG-axioms* **by** *simp*
have $\nexists v.\ v \in (verts\ G) - C$ **using** *assms* **by** *simp*
then have $(\forall v{\in}verts\ G - C.\ (\forall c{\in}C.\ c \rightarrow^+ v) \vee (\forall c{\in}C.\ v \rightarrow^+ c) \vee (\forall c{\in}C.$
$\neg\ v \rightarrow^+ c \wedge \neg\ v \rightarrow^+ c))$
by *auto*
then show $C \subseteq$ *verts* $G\ \wedge$
blockDAG $(G \upharpoonright C)\ \wedge$
$(\forall v{\in}verts\ G - C.\ (\forall c{\in}C.\ c \rightarrow^+ v) \vee (\forall c{\in}C.\ v \rightarrow^+ c) \vee (\forall c{\in}C.\ \neg\ v \rightarrow^+ c$

$\wedge \neg\ v \rightarrow^{+} c$))
    using *subset bD* **by** *simp*
  qed
qed

lemma (**in** *blockDAG*) *compositionExists*:
shows $\exists$ *C*. *composition G C*
proof
let *?C* = *verts G*
show *composition G ?C* **using** *trivialComposition* **by** *auto*
qed

lemma (**in** *blockDAG*) *compositionGraphExists*:
shows $\exists$ *G′*. *compositionGraph G G′*
proof −
obtain *C* **where** *c-def*: *C* = *verts G* **by** *auto*
then have *composition G C* **using** *trivialComposition* **by** *simp*
obtain *G′*::(*′a set*, *′b*) *pre-digraph*
where *g′-def*: *verts G′* = {*C*}
by (*metis induce-subgraph-verts*)
have *compositionGraph G G′* **unfolding** *compositionGraph-axioms-def compositionGraph-def*
g′-def c-def
proof *safe*
show *blockDAG G* **using** *blockDAG-axioms* **by** *simp*
next
show *composition G* (*verts G*) **using** *trivialComposition* **by** *simp*
next
fix *x*
assume $x \in$ *verts G*
then show $x \in \bigcup$ {*verts G*} **by** *simp*
qed
then show *?thesis* **by** *auto*
qed
end
theory *SpectreComposition*
imports *Main Graph-Theory.Graph-Theory blockDAG Composition Spectre*
begin

deprecated

end