

blockDAGs

Jörn

August 31, 2021

Contents

1	Digraph Utilities	7
2	DAG	8
2.1	Functions and Definitions	8
2.2	Lemmas	9
2.2.1	Tips	9
2.2.2	Anticone	10
2.2.3	Future Nodes	11
2.2.4	Past Nodes	11
2.2.5	Reduce Past	12
2.2.6	Reduce Past Reflexiv	13
2.2.7	Reachability cases	13
2.2.8	Soundness of the topological sort algorithm	16
3	blockDAGs	20
3.1	Functions and Definitions	20
3.2	Lemmas	20
3.2.1	Genesis	20
3.2.2	Tips	21
3.3	Future Nodes	27
3.3.1	Reduce Past	28
3.3.2	Reduce Past Reflexiv	32
3.3.3	Genesis Graph	34
4	Spectre	43
4.1	Definitions	43
4.2	Lemmas	44
5	GHOSTDAG	48
5.1	Functions and Definitions	48
5.2	Soundness	49
5.2.1	Soundness of the <i>add – set – list</i> function	49

5.2.2	Soundness of the <i>add – if – blue</i> function	50
5.2.3	Soundness of the <i>larger – blue – tuple</i> comparison	50
5.2.4	Soundness of the <i>choose_max_bblue_set</i> function	51
5.2.5	Auxiliary lemmas for OrderDAG	51
5.2.6	OrderDAG soundness	54
6	Extend blockDAGs	60
6.1	Definitions	60
6.2	Append-One Lemmas	60
6.3	Honest-Append-One Lemmas	66
6.4	Honest-Dishonest-Append-One Lemmas	67
7	SPECTRE properties	71
7.1	SPECTRE Order Preserving	71
8	Code Generation	89
8.1	Extend Graph	91
9	GHOSTDAG properties	91
9.1	GHOSTDAG Order Preserving	91
9.2	GHOSTDAG Linear Order	94
9.3	GHOSTDAG One Appending Monotone	95
9.4	GHOSTDAG Not One Appending Robust	96

```

theory Utils
  imports Main
begin

```

The following functions transform a list L to a relation containing a tuple (a, b) iff $a = b$ or a precedes b in the list L

```

fun list-to-rel:: 'a list  $\Rightarrow$  'a rel
  where list-to-rel [] = {}
  | list-to-rel (x#xs) = {x}  $\times$  (set (x#xs))  $\cup$  list-to-rel xs

```

```

lemma list-to-rel-in : (a,b)  $\in$  (list-to-rel L)  $\longrightarrow$  a  $\in$  set L  $\wedge$  b  $\in$  set L
proof(induct L, auto) qed

```

Show soundness of list-to-rel

```

lemma list-to-rel-equal:
  (a,b)  $\in$  list-to-rel L  $\longleftrightarrow$  ( $\exists k::nat.$  hd (drop k L) = a  $\wedge$  b  $\in$  set (drop k L))
proof(safe)
  assume (a, b)  $\in$  list-to-rel L
  then show  $\exists k.$  hd (drop k L) = a  $\wedge$  b  $\in$  set (drop k L)
  proof(induct L)
    case Nil
    then show ?case by auto

```

```

next
  case (Cons a2 L)
  then consider (a, b) ∈ {a2} × set (a2 # L) | (a,b) ∈ list-to-rel L by auto
  then show ?case unfolding list-to-rel.simps(2)
  proof(cases)
    case 1
    then have a = hd (a2 # L) by auto
    moreover have b ∈ set (a2 # L) using 1 by auto
    ultimately show ?thesis using drop0
      by metis
  next
  case 2
  then obtain k where k-in : hd (drop k (L)) = a ∧ b ∈ set (drop k (L))
    using Cons(1) by auto
  show ?thesis proof
    let ?k = Suc k
    show hd (drop ?k (a2 # L)) = a ∧ b ∈ set (drop ?k (a2 # L))
      unfolding drop-Suc using k-in by auto
  qed
qed
qed
next
fix k
assume b ∈ set (drop k L)
  and a = hd (drop k L)
then show (hd (drop k L), b) ∈ list-to-rel L
proof(induct L arbitrary: k)
  case Nil
  then show ?case by auto
next
case (Cons a L)
consider (zero) k = 0 | (more) k > 0 by auto
then show ?case
proof(cases)
  case zero
  then show ?thesis using Cons drop-0 by auto
next
case more
then obtain k2 where k2-in: k = Suc k2
  using gr0-implies-Suc by auto
  show ?thesis using Cons unfolding k2-in drop-Suc list-to-rel.simps(2) by
auto
qed
qed
qed

lemma list-to-rel-append:
  assumes a ∈ set L
  shows (a,b) ∈ list-to-rel (L @ [b])

```

```

using assms
proof(induct L, simp, auto) qed

```

For every distinct L, list-to-rel L return a linear order on set L

```

lemma list-order-linear:
  assumes distinct L
  shows linear-order-on (set L) (list-to-rel L)
  unfolding linear-order-on-def total-on-def partial-order-on-def preorder-on-def
refl-on-def
trans-def antisym-def
proof(safe)
  fix a b
  assume  $(a, b) \in \text{list-to-rel } L$ 
  then show  $a \in \text{set } L$ 
  proof(induct L, auto) qed
next
  fix a b
  assume  $(a, b) \in \text{list-to-rel } L$ 
  then show  $b \in \text{set } L$ 
  proof(induct L, auto) qed
next
  fix x
  assume  $x \in \text{set } L$ 
  then show  $(x, x) \in \text{list-to-rel } L$ 
  proof(induct L, auto) qed
next
  fix x y z
  assume as1:  $(x, y) \in \text{list-to-rel } L$ 
  and as2:  $(y, z) \in \text{list-to-rel } L$ 
  then show  $(x, z) \in \text{list-to-rel } L$ 
  using assms
proof(induct L)
  case Nil
  then show ?case by auto
next
  case (Cons a L)
  then consider (nor)  $(x, y) \in \{a\} \times \text{set } (a \# L) \wedge (y, z) \in \{a\} \times \text{set } (a \# L)$ 
    |  $(xy) (x, y) \in \text{list-to-rel } L \wedge (y, z) \in \{a\} \times \text{set } (a \# L)$ 
    |  $(yz) (y, z) \in \text{list-to-rel } L \wedge (x, y) \in \{a\} \times \text{set } (a \# L)$ 
    | (both)  $(y, z) \in \text{list-to-rel } L \wedge (x, y) \in \text{list-to-rel } L$  by auto
  then show ?case proof(cases)
    case nor
    then show ?thesis by auto
next
  case xy
  then have  $y \in \text{set } L$  using list-to-rel-in by metis
  also have  $y = a$  using xy by auto
  ultimately have  $\neg \text{distinct } (a \# L)$ 
  by simp

```

```

    then show ?thesis using Cons by auto
  next
    case yz
    then show ?thesis using list-to-rel.simps(2)
      by (metis Cons.prem(2) SigmaD1 SigmaI UnI1 list-to-rel-in)
  next
    case both
    then show ?thesis unfolding list-to-rel.simps(2) using Cons by auto
  qed
qed
next
  fix x y
  assume (x, y) ∈ list-to-rel L
  and (y, x) ∈ list-to-rel L
  then show x = y
    using assms
  proof(induct L, simp)
    case (Cons a L)
    then consider (nor) (x, y) ∈ {a} × set (a # L) ∧ (y, x) ∈ {a} × set (a # L)
      | (xy) (x,y) ∈ list-to-rel L ∧ (y, x) ∈ {a} × set (a # L)
      | (yz) (y,x) ∈ list-to-rel L ∧ (x, y) ∈ {a} × set (a # L)
      | (both) (y,x) ∈ list-to-rel L ∧ (x,y) ∈ list-to-rel L by auto
    then show ?case unfolding list-to-rel.simps
  proof(cases)
    case nor
    then show ?thesis by auto
  next
    case xy
    then show ?thesis
      by (metis Cons.prem(3) SigmaD1 distinct.simps(2) list-to-rel-in singletonD)
  next
    case yz
    then show ?thesis
      by (metis Cons.prem(3) SigmaD1 distinct.simps(2) list-to-rel-in singletonD)
  next
    case both
    then show ?thesis using Cons by auto
  qed
qed
next
  fix x y
  assume x ∈ set L
  and y ∈ set L
  and x ≠ y
  and (y, x) ∉ list-to-rel L
  then show (x, y) ∈ list-to-rel L
  proof(induct L, auto) qed

```

qed

```
lemma list-to-rel-mono:
  assumes  $(a,b) \in \text{list-to-rel } (L)$ 
  shows  $(a,b) \in \text{list-to-rel } (L @ L2)$ 
  using assms
proof(induct L2 arbitrary: L, simp)
  case (Cons a L2)
  then show ?case
  proof(induct L, auto)
    qed
  qed
qed
```

```
lemma list-to-rel-mono2:
  assumes  $(a,b) \in \text{list-to-rel } (L2)$ 
  shows  $(a,b) \in \text{list-to-rel } (L @ L2)$ 
  using assms
proof(induct L2 arbitrary: L, simp)
  case (Cons a L2)
  then show ?case
  proof(induct L, auto)
    qed
  qed
qed
```

```
lemma map-snd-map:  $\bigwedge L. (\text{map snd } (\text{map } (\lambda i. (P\ i, i))\ L)) = L$ 
proof -
  fix L
  show  $\text{map snd } (\text{map } (\lambda i. (P\ i, i))\ L) = L$ 
  proof(induct L)
    case Nil
    then show ?case by auto
  next
    case (Cons a L)
    then show ?case by auto
  qed
qed
end
```

```
theory DigraphUtils
  imports Main Graph-Theory.Graph-Theory
begin
```

1 Digraph Utilities

lemma *graph-equality*:

assumes *digraph* $G \wedge$ *digraph* C
assumes $\text{verts } G = \text{verts } C \wedge \text{arcs } G = \text{arcs } C \wedge \text{head } G = \text{head } C \wedge \text{tail } G = \text{tail } C$
shows $G = C$
by (*simp add: assms(2)*)

lemma (*in digraph*) *del-vert-not-in-graph*:

assumes $b \notin \text{verts } G$
shows $(\text{pre-digraph.del-vert } G \ b) = G$
proof –
have $v: \text{verts } (\text{pre-digraph.del-vert } G \ b) = \text{verts } G$
using *assms(1)*
by (*simp add: pre-digraph.verts-del-vert*)
have $\forall e \in \text{arcs } G. \text{tail } G \ e \neq b \wedge \text{head } G \ e \neq b$ **using** *digraph-axioms*
assms digraph.axioms(2) loopfree-digraph.axioms(1)
by *auto*
then have $\text{arcs } G \subseteq \text{arcs } (\text{pre-digraph.del-vert } G \ b)$
using *assms*
by (*simp add: pre-digraph.arcs-del-vert subsetI*)
then have $e: \text{arcs } G = \text{arcs } (\text{pre-digraph.del-vert } G \ b)$
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* $G \wedge$ *digraph* H
shows *subgraph* $(\text{pre-digraph.del-arc } H \ e2) (\text{pre-digraph.del-arc } G \ e2)$
using *subgraph-def pre-digraph.del-arc-simps Diff-iff*
proof –
have $f1: \forall p \text{ pa}. \text{subgraph } p \text{ pa} = ((\text{verts } p::'a \text{ set}) \subseteq \text{verts } \text{pa} \wedge (\text{arcs } p::'b \text{ set}) \subseteq \text{arcs } \text{pa} \wedge$
 $\text{wf-digraph } \text{pa} \wedge \text{wf-digraph } p \wedge \text{compatible } \text{pa } p)$
using *subgraph-def* **by** *blast*
have $\text{arcs } H - \{e2\} \subseteq \text{arcs } G - \{e2\}$ **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:  $\bigwedge V. (\text{digraph } V \implies \text{card } (\text{verts } V) = 0 \implies P \ V)$ 
 $\bigwedge W \ c. (\bigwedge V. (\text{digraph } V \implies \text{card } (\text{verts } V) = c \implies P \ V))$ 
 $\implies (\text{digraph } W \implies \text{card } (\text{verts } W) = (\text{Suc } c) \implies P \ W)$ 
shows  $\bigwedge Z. \text{digraph } Z \implies P \ Z$ 
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  $(\bigwedge Z. x = \text{card } (\text{verts } Z) \implies \text{digraph } Z \implies P \ Z)$ 
    show ?case
    by (metis local.cases(2) su.hyps(1) su.hyps(2) su.premis)
  qed
qed
end

```

```

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

```

2 DAG

```

locale DAG = digraph +
  assumes cycle-free:  $\neg(v \rightarrow^+_G v)$ 

```

```

sublocale DAG  $\subseteq$  wf-digraph using DAG-def digraph-def nomulti-digraph-def
DAG-axioms by auto

```

2.1 Functions and Definitions

```

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

```

```

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

```

```

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

```

```

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

```



```

fun anticone:: ('a,'b) pre-digraph  $\Rightarrow$  'a  $\Rightarrow$  'a set
  where anticone G a = {b  $\in$  verts G.  $\neg$ (a  $\rightarrow^+_G$  b  $\vee$  b  $\rightarrow^+_G$  a  $\vee$  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)  $\wedge$  ( $\forall$  x  $\in$  verts G.  $\neg$  x  $\rightarrow^+_G$  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)

```

2.2 Lemmas

```

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

```

2.2.1 Tips

```

lemma (in wf-digraph) tips-not-referenced:
  assumes is-tip G t
  shows  $\forall$  x.  $\neg$  x  $\rightarrow^+_G$  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  $\rightarrow^+_G$  del-vert t v
  then have v  $\rightarrow^+_G$  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  $x \in \textit{tips } G$ 
  shows is-tip G x using tips-def CollectD assms(1) by metis

```

```

lemma tip-in-verts:
  assumes  $x \in \textit{tips } G$ 
  shows  $x \in \textit{verts } G$  using tips-def CollectD assms(1) by metis

```

2.2.2 Anticone

```

lemma (in DAG) tips-anticone:
  assumes  $a \in \textit{tips } G$ 
  and  $b \in \textit{tips } G$ 
  and  $a \neq b$ 
  shows  $a \in \textit{anticone } G \ b$ 
proof(rule ccontr)
  assume  $a \notin \textit{anticone } G \ b$ 
  then have  $k: (a \rightarrow^+ b \vee b \rightarrow^+ a \vee a = b)$  using anticone.simps assms tips-def
  by fastforce
  then have  $\neg (\forall x \in \textit{verts } G. \ x \rightarrow^+ a) \vee \neg (\forall x \in \textit{verts } G. \ x \rightarrow^+ b)$  using
reachable1-in-verts
  assms(3) cycle-free
  by (metis)
  then have  $\neg \textit{is-tip } G \ a \vee \neg \textit{is-tip } G \ b$  using assms(3) is-tip.simps k
  by (metis)
  then have  $\neg a \in \textit{tips } G \vee \neg b \in \textit{tips } G$  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 $a \notin (\text{anticone } G \ a)$ **by** *auto*

2.2.3 Future Nodes

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

2.2.4 Past Nodes

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

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

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

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

lemma (in *DAG*) *finite-past: finite* ($\text{past-nodes } G \ a$)
by (*metis finite-verts rev-finite-subset past-nodes-verts*)

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

lemma (in *DAG*) *finite-future: finite* ($\text{future-nodes } G \ a$)
by (*metis finite-verts rev-finite-subset future-nodes-verts*)

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

2.2.5 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*:
 $e \in \text{arcs } (\text{reduce-past } G \ a) \implies e \in \text{arcs } G$
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 $u \rightarrow^+_{\text{reduce-past } G \ a} v$
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 $u \rightarrow^+_G v$
and $u \in \text{past-nodes } G \ a$
and $v \in \text{past-nodes } G \ a$
shows $u \rightarrow^+_{\text{reduce-past } G \ a} v$
using *assms*

proof(*induct* *u v*)

case (*r-into-trancl* *u v*)

then obtain *e* **where** *e-in*: $\text{arc } e \ (u, v)$ **using** *arc-def DAG-axioms wf-digraph-def*
by *auto*

then have *e-in2*: $e \in \text{arcs } (\text{reduce-past } G \ a)$ **unfolding** *reduce-past.simps induce-subgraph-arcs*

using *arcE r-into-trancl.prem(1) r-into-trancl.prem(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 $u \rightarrow_{\text{reduce-past } G \ a} v$ **using** *e-in2 wf-digraph.dominatesI DAG-axioms*

by (*metis reduce-past.simps wellformed-induce-subgraph*)

then show *?case* **by** *auto*

```

next
case (tranc1-into-tranc1 a2 b c)
then have b-in:  $b \in \text{past-nodes } G \ a$  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:  $a2 \rightarrow^+ \text{reduce-past } G \ a \ b$  using tranc1-into-tranc1 by auto
then obtain e where e-in:  $\text{arc } e \ (b,c)$  using tranc1-into-tranc1
  arc-def DAG-axioms wf-digraph-def by auto
then have e-in2:  $e \in \text{arcs } (\text{reduce-past } G \ a)$  unfolding reduce-past.simps induce-subgraph-arcs
  using arcE tranc1-into-tranc1
  b-in by blast
then have arc-to-ends ( $\text{reduce-past } G \ a$ )  $e = (b,c)$  unfolding reduce-past.simps
using e-in
  arcE arc-to-ends-def induce-subgraph-head induce-subgraph-tail
  by metis
then have  $b \rightarrow \text{reduce-past } G \ a \ c$  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 tranc1.tranc1-into-tranc1)
qed

```

lemma (in *DAG*) *reduce-past-pathr*:
 assumes $u \rightarrow^* \text{reduce-past } G \ a \ v$
 shows $u \rightarrow^* G \ v$
by (meson *assms induced-subgraph-altdef reachable-mono reduce-past-induced-subgraph*)

2.2.6 Reduce Past Reflexiv

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

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

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

2.2.7 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 ∈ tips G
  shows verts G = {x} ∪ (anticone G x) ∪ (verts (reduce-past G x))
proof
  show verts G ⊆ {x} ∪ anticone G x ∪ verts (reduce-past G x)
  proof(rule subsetI)
    fix xa
    assume in-V: xa ∈ verts G
    then show xa ∈ {x} ∪ anticone G x ∪ 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} ∪ anticone G x ∪ verts (reduce-past G x) ⊆ verts G using di-
graph.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-comp2:
  assumes x ∈ tips G
  and a ∈ verts G
  obtains a = x
  | a ∈ anticone G x
  | a ∈ past-nodes G x
  using assms
proof(cases a x rule:reachable1-cases)
  case one
  then show ?thesis

```

```

    by (metis assms(1) tips-not-referenced tips-tips)
next
  case two
  then show ?thesis using past-nodes.simps wf-digraph.reachable1-in-verts(2) wf-digraph-axioms
    mem-Collect-eq that(3)
    by (metis (no-types, lifting))
next
  case nR
  then show ?thesis using that(2) anticone.simps assms by auto
qed

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

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

end
theory TopSort
  imports DAGs Utils
begin

```

Function to sort a list L under a graph G such if a references b , b precedes

a in the list

```
fun top-insert:: ('a::linorder,'b) pre-digraph  $\Rightarrow$  'a list  $\Rightarrow$  'a  $\Rightarrow$  'a list
  where top-insert  $G \ [] \ a = [a]$ 
    | top-insert  $G \ (b \# L) \ a = (if \ (b \rightarrow^+_G \ a) \ then \ (a \# \ (b \# L)) \ else \ (b \#$ 
top-insert  $G \ L \ a))$ 
```

```
fun top-sort:: ('a::linorder,'b) pre-digraph  $\Rightarrow$  'a list  $\Rightarrow$  'a list
  where top-sort  $G \ [] = []$ 
    | top-sort  $G \ (a \# L) = top-insert \ G \ (top-sort \ G \ L) \ a$ 
```

2.2.8 Soundness of the topological sort algorithm

lemma top-insert-set: $set \ (top-insert \ G \ L \ a) = set \ L \cup \{a\}$

proof(*induct L, simp-all, auto*) **qed**

lemma top-sort-con: $set \ (top-sort \ G \ L) = set \ L$

proof(*induct L*)

case Nil

then show ?case by auto

next

case (Cons a L)

then show ?case using top-sort.simps(2) top-insert-set insert-is-Un list.simps(15)
sup-commute

by (metis)

qed

lemma top-insert-len: $length \ (top-insert \ G \ L \ a) = Suc \ (length \ L)$

proof(*induct L*)

case Nil

then show ?case by auto

next

case (Cons a L)

then show ?case using top-insert.simps(2) by auto

qed

lemma top-sort-len: $length \ (top-sort \ G \ L) = length \ L$

proof(*induct L, simp*)

case (Cons a L)

then have $length \ (a \# L) = Suc \ (length \ L)$ by auto

then show ?case using

top-insert-len top-sort.simps(2) Cons

by (simp add: top-insert-len)

qed

lemma top-insert-mono:

assumes $(y, x) \in list-to-rel \ ls$

shows $(y, x) \in list-to-rel \ (top-insert \ G \ ls \ l)$

using *assms*


```

proof(induct ls, simp)
  case (Cons a ls)
  consider (rec)  $a \rightarrow^+_G l \mid (nrec) \neg a \rightarrow^+_G l$  by auto
  then show ?case
  proof(cases)
    case rec
    then have sinse: (top-insert  $G (a \# ls) l$ ) =  $l \# a \# ls$ 
      unfolding top-insert.simps by simp
    show ?thesis unfolding sinse list-to-rel.simps using Cons
      by auto
  next
    case nrec
    then have sinse: (top-insert  $G (a \# ls) l$ ) =  $a \# \text{top-insert } G \text{ } ls \text{ } l$ 
      unfolding top-insert.simps by simp
    consider (ya)  $y = a \mid (yan) (y, x) \in \text{list-to-rel } ls$  using Cons by auto
    then show ?thesis proof(cases)
      case ya
      then show ?thesis unfolding sinse list-to-rel.simps
        by (metis Cons.premis SigmaI UnI1 top-insert-set insertCI list-to-rel-in sinse)
      case yan
      then show ?thesis using Cons unfolding sinse list-to-rel.simps by auto
    qed
  qed
qed

lemma top-sort-mono:
  assumes  $(y, x) \in \text{list-to-rel } (\text{top-sort } G \text{ } ls)$ 
  shows  $(y, x) \in \text{list-to-rel } (\text{top-sort } G (l \# ls))$ 
  using assms
  by (simp add: top-insert-mono)

fun (in DAG) top-sorted :: 'a list  $\Rightarrow$  bool where
  top-sorted [] = True |
  top-sorted ( $x \# ys$ ) =  $((\forall y \in \text{set } ys. \neg x \rightarrow^+_G y) \wedge \text{top-sorted } ys)$ 

lemma (in DAG) top-sorted-sub:
  assumes  $S = \text{drop } k \text{ } L$ 
  and top-sorted  $L$ 
  shows top-sorted  $S$ 
  using assms
proof(induct k arbitrary: L S)
  case 0
  then show ?case by auto
next
  case (Suc k)

```

```

    then show ?case unfolding drop-Suc using top-sorted.simps
    by (metis Suc.premis(1) drop-Nil list.sel(3) top-sorted.elims(2))
qed

```

```

lemma top-insert-part-ord:
  assumes DAG G
  and DAG.top-sorted G L
  shows DAG.top-sorted G (top-insert G L a)
  using assms
proof(induct L)
  case Nil
  then show ?case
    by (simp add: DAG.top-sorted.simps)
next
  case (Cons b list)
  consider (re)  $b \rightarrow^+_G a \mid (nre) \neg b \rightarrow^+_G a$  by auto
  then show ?case proof(cases)
    case re
    have  $(\forall y \in \text{set } (b \# \text{list}). \neg a \rightarrow^+_G y)$ 
    proof(rule ccontr)
      assume  $\neg (\forall y \in \text{set } (b \# \text{list}). \neg a \rightarrow^+_G y)$ 
      then obtain wit where wit-in:  $wit \in \text{set } (b \# \text{list}) \wedge a \rightarrow^+_G wit$  by auto
      then have  $b \rightarrow^+_G wit$  using re
      by auto
      then have  $\neg \text{DAG.top-sorted } G \ (b \# \text{list})$ 
      using wit-in using DAG.top-sorted.simps(2) Cons(2)
      by (metis DAG.cycle-free set-ConsD)
      then show False using Cons by auto
    qed
    then show ?thesis using assms(1) DAG.top-sorted.simps Cons
    by (simp add: DAG.top-sorted.simps(2) re)
  next
    case nre
    have DAG.top-sorted G list using Cons(2,3)
    by (metis DAG.top-sorted.simps(2))
    then have DAG.top-sorted G (top-insert G list a)
    using Cons(1,2) by auto
    moreover have  $(\forall y \in \text{set } (\text{top-insert } G \text{ list } a). \neg b \rightarrow^+_G y)$  using top-insert-set
    Cons DAG.top-sorted.simps(2) nre
    by (metis Un-iff empty-iff empty-set list.simps(15) set-ConsD)
    ultimately show ?thesis using Cons(2)
    by (simp add: DAG.top-sorted.simps(2) nre)
  qed
qed

```

```

lemma top-sort-sorted:

```

```

    assumes DAG G
    shows DAG.top-sorted G (top-sort G L)
    using assms
  proof(induct L)
    case Nil
    then show ?case
      by (simp add: DAG.top-sorted.simps(1))
    case (Cons a L)
    then show ?case unfolding top-sort.simps using top-insert-part-ord by auto
  qed

lemma top-sorted-rel:
  assumes DAG G
  and  $y \rightarrow^+_G x$ 
  and  $x \in \text{set } L$ 
  and  $y \in \text{set } L$ 
  and DAG.top-sorted G L
  shows  $(x,y) \in \text{list-to-rel } L$ 
  using assms
  proof(induct L, simp)
    have une:  $x \neq y$  using assms
    by (metis DAG.cycle-free)
    case (Cons a L)
    then consider  $x = a \wedge y \in \text{set } (a \# L) \mid y = a \wedge x \in \text{set } L \mid x \in \text{set } L \wedge y \in$ 
    set L
    using une by auto
    then show ?case proof(cases)
      case 1
      then show ?thesis unfolding list-to-rel.simps by auto
    next
      case 2
      then have  $\neg \text{DAG.top-sorted } G (a \# L)$ 
      using assms DAG.top-sorted.simps(2)
      by fastforce
      then show ?thesis using Cons by auto
    next
      case 3
      then show ?thesis unfolding list-to-rel.simps using Cons DAG.top-sorted.simps(2)
    Un-iff
    by metis
  qed
qed

lemma top-sort-rel:
  assumes DAG G
  and  $y \rightarrow^+_G x$ 
  and  $x \in \text{set } L$ 
  and  $y \in \text{set } L$ 
  shows  $(x,y) \in \text{list-to-rel } (\text{top-sort } G L)$ 

```

```

    using assms top-sort-sorted top-sorted-rel top-sort-con
    by metis

```

```

end

```

```

theory blockDAG
  imports DAGs DigraphUtils
begin

```

3 blockDAGs

```

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

```

```

lemma bD: blockDAG G using blockDAG-axioms by simp

```

```

end

```

3.1 Functions and Definitions

```

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

```

```

definition (in blockDAG) genesis-node:: 'a
  where genesis-node =  $(\text{THE } x. \text{is-genesis-node } x)$ 

```

3.2 Lemmas

```

lemma subs:
  assumes blockDAG G
  shows  $\text{DAG } G \wedge \text{digraph } G \wedge \text{fin-digraph } G \wedge \text{wf-digraph } G$ 
  using assms blockDAG-def DAG-def digraph-def fin-digraph-def by blast

```

3.2.1 Genesis

```

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

```

```

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

```

```

lemma (in blockDAG) unique-genesis:  $\text{is-genesis-node } a \wedge \text{is-genesis-node } b \longrightarrow a = b$ 

```

```

using genesisAlt reachable-trans cycle-free
    reachable-refl reachable-reachable1-trans reachable-neq-reachable1
by (metis (full-types))

```

```

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

```

```

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

```

```

lemma (in blockDAG) genesis-reaches-nothing:
  assumes  $a \rightarrow^+ b$ 
  shows  $\neg \text{is-genesis-node } a$ 
using is-genesis-node.simps genesis-node-def genesis-existAlt cycle-free
    reachable1-reachable-trans assms reachable1-in-verts(2)
by (metis)

```

3.2.2 Tips

```

lemma (in blockDAG) tips-exist:
   $\exists x. \text{is-tip } G \ x$ 
  unfolding is-tip.simps
proof (rule ccontr)
  assume  $\nexists x. x \in \text{verts } G \wedge (\forall xa \in \text{verts } G. (xa, x) \notin (\text{arcs-ends } G)^+)$ 
  then have contr:  $\forall x. x \in \text{verts } G \longrightarrow (\exists y. y \rightarrow^+ x)$ 
    by auto
  have  $\forall x y. y \rightarrow^+ x \longrightarrow \{z. x \rightarrow^+ z\} \subseteq \{z. y \rightarrow^+ z\}$ 
    using Collect-mono transcl-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:  $\forall x. \{z. x \rightarrow^+ z\} \subseteq \text{verts } G$ 
    using reachable1-in-verts by auto
  then have fin:  $\forall x. \text{finite } \{z. x \rightarrow^+ z\}$ 
    using finite-verts finite-subset
    by metis
  then have trans:  $\forall x y. y \rightarrow^+ x \longrightarrow \text{card } \{z. x \rightarrow^+ z\} < \text{card } \{z. y \rightarrow^+ z\}$ 
    using sub psubset-card-mono by metis
  then have inf:  $\forall y \in \text{verts } G. \exists x. \text{card } \{z. x \rightarrow^+ z\} > \text{card } \{z. y \rightarrow^+ z\}$ 
    using fin contr genesis
    reachable1-in-verts(1)
    by (metis (mono-tags, lifting))
  have all:  $\forall k. \exists x \in \text{verts } G. \text{card } \{z. x \rightarrow^+ z\} > k$ 
proof
  fix  $k$ 
  show  $\exists x \in \text{verts } G. k < \text{card } \{z. x \rightarrow^+ z\}$ 

```

```

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:  $\exists x \in \text{verts } G. \text{ card } (\text{verts } G) < \text{ card } \{z. x \rightarrow^+ z\}$  by simp
also
have  $\forall x. \text{ card } \{z. x \rightarrow^+ z\} \leq \text{ card } (\text{verts } G)$ 
  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  $\neq \{\}$ 
proof(rule ccontr)
  assume as1:  $\neg \text{ tips } G \neq \{\}$ 
  obtain t where t-in: is-tip G t using tips-exist by auto
  then have t-inV:  $t \in \text{verts } G$  by auto
  then have  $t \in \text{tips } G$  using tips-def CollectI t-in by metis
  then show False using as1 by auto
qed

```

```

lemma (in blockDAG) reached-by:
  assumes  $v \notin \text{tips } G$ 
  and  $v \in \text{verts } G$ 
  shows  $\exists t \in \text{tips } G. t \rightarrow^+ v$ 
  using assms
  unfolding tips-def is-tip.simps
  by auto

```

```

lemma (in blockDAG) reached-by-tip:
  assumes  $v \notin \text{tips } G$ 
  and  $v \in \text{verts } G$ 
  shows  $\exists t \in \text{tips } G. t \rightarrow^+ v$ 
proof(rule ccontr)
  assume as1:  $\neg (\exists t \in \text{tips } G. t \rightarrow^+ v)$ 
  then have  $\forall w. w \rightarrow^+ v \longrightarrow \neg \text{ is-tip } G w$ 
  unfolding tips-def using reachable1-in-verts(1) CollectI
  by blast

```

```

then have contr:  $\forall w. w \rightarrow^+ v \longrightarrow (\exists y. y \rightarrow^+ w \wedge y \rightarrow^+ v)$ 
  using as1 reachable1-in-verts(1) reached-by trancl-trans
  by metis
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:  $\forall x. \{z. x \rightarrow^+ z\} \subseteq \text{verts } G$ 
  using reachable1-in-verts by auto
then have fin:  $\forall x. \text{finite } \{z. x \rightarrow^+ z\}$ 
  using finite-verts finite-subset
  by metis
then have trans:  $\forall x y. y \rightarrow^+ x \longrightarrow \text{card } \{z. x \rightarrow^+ z\} < \text{card } \{z. y \rightarrow^+ z\}$ 
  using sub psubset-card-mono by metis
then have inf:  $\forall w. w \rightarrow^+ v \longrightarrow (\exists x. x \rightarrow^+ v \wedge \text{card } \{z. x \rightarrow^+ z\} > \text{card } \{z.$ 
 $w \rightarrow^+ z\})$ 
  using fin contr genesis
  reachable1-in-verts(1)
  by metis
have all:  $\forall k. \exists w \in \text{verts } G. w \rightarrow^+ v \wedge \text{card } \{z. w \rightarrow^+ z\} > k$ 
proof
  fix k
  show  $\exists w \in \text{verts } G. w \rightarrow^+ v \wedge \text{card } \{z. w \rightarrow^+ z\} > k$ 
  proof(induct k )
    case 0
    then show ?case
      using inf neq0-conv assms(1) assms(2) local.trans reached-by contr
      by (metis less-nat-zero-code)
    next
    case (Suc k)
    then show ?case
      using Suc-lessI inf reachable1-in-verts(1)
      by (metis)
    qed
  qed
then have less:  $\exists x \in \text{verts } G. \text{card } (\text{verts } G) < \text{card } \{z. x \rightarrow^+ z\}$ 
  by blast
also
have  $\forall x. \text{card } \{z. x \rightarrow^+ z\} \leq \text{card } (\text{verts } G)$ 
  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-unequal-gen:
  assumes card( verts G) > 1
  and is-tip G p

```

```

shows  $\neg$  is-genesis-node p
proof (rule ccontr)
  assume as:  $\neg \neg$  is-genesis-node p
  have b1:  $1 < \text{card } (\text{verts } G)$  using assms by linarith
  then have  $0 < \text{card } ((\text{verts } G) - \{p\})$  using card-Suc-Diff1 as finite-verts b1
by auto
  then have  $((\text{verts } G) - \{p\}) \neq \{\}$  using card-gt-0-iff by blast
  then obtain y where y-def:  $y \in (\text{verts } G) - \{p\}$  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  $\exists p. p \in \text{verts } G \wedge \text{is-tip } G p \wedge \neg \text{is-genesis-node } p$ 
proof -
  have b1:  $1 < \text{card } (\text{verts } G)$  using assms by linarith
  obtain x where x-in:  $x \in (\text{verts } G) \wedge \text{is-genesis-node } x$ 
    using genesis genesisAlt genesis-node-def by blast
  then have  $0 < \text{card } ((\text{verts } G) - \{x\})$  using card-Suc-Diff1 x-in finite-verts b1
by auto
  then have  $((\text{verts } G) - \{x\}) \neq \{\}$  using card-gt-0-iff by blast
  then obtain y where y-def:  $y \in (\text{verts } G) - \{x\}$  by auto
  then have uneq:  $y \neq x$  by auto
  have y-in:  $y \in (\text{verts } G)$  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:  $z \in (\text{verts } G) - \{x\} \wedge \text{is-tip } G z$  using tips-exist
    is-tip.simps by auto
  then have uneq:  $z \neq x$  by auto
  have z-in:  $z \in \text{verts } G$  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 ¬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:  $t \notin \text{set } (\text{map } (\text{tail } G) p)$ 
proof(rule ccontr)
  assume as2:  $\neg t \notin \text{set } (\text{map } (\text{tail } G) p)$ 
  then have tl2:  $t \in \text{set } (\text{map } (\text{tail } G) p)$  by auto
  then have  $t \in \text{set } (\text{map } (\text{head } G) p)$ 
  proof (induct rule: cas.induct)
    case (1 u v)
    then have  $v \notin \text{set } (\text{map } (\text{tail } G) [])$  by auto
    then show  $v \in \text{set } (\text{map } (\text{tail } G) []) \implies v \in \text{set } (\text{map } (\text{head } G) [])$ 
      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  $r \in \text{verts } (\text{del-vert } t)$  using in-del by simp
next
  fix x
  assume as3:  $x \in \text{set } p$ 
  then have ht:  $\text{head } G \ x \neq t \wedge \text{tail } G \ x \neq t$ 
    using no-head no-tail by auto
  have  $x \in \text{arcs } G$ 
    using awalk-def path subsetD as3 by auto
  then show  $x \in \text{arcs } (\text{del-vert } t)$  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:  $\bigwedge r. (\text{cas } r \ p \ g \implies \text{pre-digraph.cas } (\text{del-vert } t) \ r \ p \ g)$ 
      and one:  $\text{cas } r \ (a \ \# \ p) \ g$ 
    then have two:  $\text{cas } (\text{head } G \ a) \ p \ g$ 
      using awalk-def by auto
    then have t:  $\text{tail } (\text{del-vert } t) \ a = r$ 
      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
  then show  $r \rightarrow^*_{\text{del-vert } t} g$  by (meson wf-digraph.reachable-awalkI
    del-tips-dag assms(1) DAG-def digraph-def fin-digraph-def)
  qed
  then show  $\exists p \in \text{verts } (\text{del-vert } t)$  .
    ( $\forall r. r \in \text{verts } (\text{del-vert } t) \longrightarrow (r \rightarrow^+_{\text{del-vert } t} p \vee r = p)$ )
    using gen genp
    by (metis reachable-rtranciI rtranciD)
  qed

```

lemma (in blockDAG) tips-cases [consumes 2, case-names ma past nma]:

```

  assumes  $p \in \text{tips } G$ 
  and  $x \in \text{verts } G$ 
  obtains (ma)  $x = p$ 
  | (past)  $x \in \text{past-nodes } G p$ 
  | (nma)  $x \in \text{anticone } G p$ 
  proof -
    consider (eq)  $x = p$  | (neg)  $\neg x = p$  by auto
    then show ?thesis
    proof (cases)
      case eq
      then show thesis using eq ma by simp
    next
      case neg
      consider (in-p)  $x \in \text{past-nodes } G p$  | (nin-p)  $x \notin \text{past-nodes } G p$  by auto
      then show ?thesis
      proof (cases)
        case in-p
        then show ?thesis using past by auto
      next
        case nin-p
        then have nn:  $\neg p \rightarrow^+_{\text{G}} x$  using nin-p past-nodes.simps assms(2) by auto
        have  $\neg x \rightarrow^+_{\text{G}} p$  using is-tip.simps assms tips-def CollectD by metis
        then have  $x \in \text{anticone } G p$  using anticone.simps neg 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  $a \in \text{verts } G$ 
  shows  $a \notin \text{future-nodes } G a$ 
  using cycle-free future-nodes.simps reachable-def by auto

```

3.3.1 Reduce Past

lemma (in *blockDAG*) *reduce-past-not-empty*:

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

lemma (in *blockDAG*) *reduce-less*:

assumes $a \in \text{verts } G$
 shows $\text{card } (\text{verts } (\text{reduce-past } G \ a)) < \text{card } (\text{verts } G)$
proof –
 have $\text{past-nodes } G \ a \subset \text{verts } G$
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 $a \in \text{verts } G$
 and $\neg \text{is-genesis-node } a$
 shows *blockDAG* $(\text{reduce-past } G \ a)$
 unfolding *blockDAG-def DAG-def blockDAG-def*
proof *safe*
 show *digraph* $(\text{reduce-past } G \ a)$
using *digraphI-induced reduce-past-induced-subgraph* **by** *auto*
next
 show *DAG-axioms* $(\text{reduce-past } G \ a)$
 unfolding *DAG-axioms-def*
using *cycle-free reduce-past-path* **by** *metis*
next
 show *blockDAG-axioms* $(\text{reduce-past } G \ a)$
 unfolding *blockDAG-axioms-def*
proof *safe*
 fix $u \ v \ e$
 assume $\text{arc: wf-digraph.arc } (\text{reduce-past } G \ a) \ e \ (u, v)$
 then show $u \rightarrow^+ \text{pre-digraph.del-arc } (\text{reduce-past } G \ a) \ e \ v \implies \text{False}$

```

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  $e \in \text{arcs (reduce-past G a)} \wedge \text{tail (reduce-past G a) } e = u$ 
     $\wedge \text{head (reduce-past G a) } e = v$ 
    using arc wf-digraph.arcE
    by metis
  then show ?thesis
    using arc-def reduce-past.simps by auto
qed
then have  $\neg u \rightarrow^+ \text{del-arc } e \ v$ 
  using only-new by auto
then show  $u \rightarrow^+ \text{pre-digraph.del-arc (reduce-past G a) } e \ v \implies \text{False}$ 
  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  $pe: p \in \text{verts (reduce-past G a)} \wedge (\forall r. r \in \text{verts (reduce-past G a)} \longrightarrow r$ 
 $\rightarrow^* \text{reduce-past G a } p)$ 
  proof
    show  $p \in \text{verts (reduce-past G a)}$  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  $\forall r. r \in \text{verts (reduce-past G a)} \longrightarrow r \rightarrow^* \text{reduce-past G a } p$ 
  proof safe
    fix  $r \ a$ 
    assume in-past:  $r \in \text{verts (reduce-past G a)}$ 
    then have con:  $r \rightarrow^* p$  using gen genesisAlt past-nodes-verts by auto
    then show  $r \rightarrow^* \text{reduce-past G a } p$ 
    proof –
      have f1:  $r \in \text{verts G} \wedge a \rightarrow^+ r$ 
      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  $r \rightarrow^* \text{aaa (past-nodes G a) (Collect (reachable G r))}$ 

```

```

       $\longrightarrow a \rightarrow^+ \text{aaa } (\text{past-nodes } G \ a) \ (\text{Collect } (\text{reachable } G \ r))$ 
    using f1 by (meson reachable1-reachable-trans)
    then have  $\text{aaa } (\text{past-nodes } G \ a) \ (\text{Collect } (\text{reachable } G \ r)) \notin \text{Collect}$ 
       $(\text{reachable } G \ r)$ 
       $\vee \text{aaa } (\text{past-nodes } G \ a) \ (\text{Collect } (\text{reachable } G \ r)) \in \text{past-nodes}$ 
    G a
    by (simp add: reachable-in-verts(2))
    then have  $\text{Collect } (\text{reachable } G \ r) \subseteq \text{past-nodes } G \ a$ 
    using f2 by (metis subsetI)
    then show ?thesis
    using con induce-reachable-preserves-paths reachable-induce-ss re-
duce-past.simps
    by (metis (no-types))
  qed
qed
qed
show
   $\exists p \in \text{verts } (\text{reduce-past } G \ a). (\forall r. r \in \text{verts } (\text{reduce-past } G \ a)$ 
     $\longrightarrow (r \rightarrow^+ \text{reduce-past } G \ a \ p \vee r = p))$ 
  using pe
  by (metis reachable-rtranclI rtranclD)
qed
qed

```

lemma (in *blockDAG*) *reduce-past-gen*:

```

  assumes  $\neg \text{is-genesis-node } a$ 
  and  $a \in \text{verts } G$ 
  shows  $\text{blockDAG.is-genesis-node } G \ b \implies \text{blockDAG.is-genesis-node } (\text{reduce-past } G \ a) \ b$ 
  proof -
    assume gen:  $\text{blockDAG.is-genesis-node } G \ b$ 
    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  $b \in (\text{past-nodes } G \ a)$  using past-nodes.simps gen by auto
    then have inv:  $b \in \text{verts } (\text{reduce-past } G \ a)$  using reduce-past.simps induce-subgraph-verts

    by auto
  have  $\forall r. r \in \text{verts } (\text{reduce-past } G \ a) \longrightarrow r \rightarrow^* \text{reduce-past } G \ a \ b$ 
  proof safe
    fix r a
    assume in-past:  $r \in \text{verts } (\text{reduce-past } G \ a)$ 
    then have con:  $r \rightarrow^* b$  using gen genesisAlt past-nodes-verts by auto
    then show  $r \rightarrow^* \text{reduce-past } G \ a \ b$ 
    proof -
      have f1:  $r \in \text{verts } G \wedge a \rightarrow^+ r$ 

```

```

    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  $r \rightarrow^* aaa\ (past-nodes\ G\ a)\ (Collect\ (reachable\ G\ r))$ 
     $\rightarrow a \rightarrow^+ aaa\ (past-nodes\ G\ a)\ (Collect\ (reachable\ G\ r))$ 
    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
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  $\neg is-genesis-node\ a$ 
  and  $a \in verts\ G$ 
  shows  $blockDAG.is-genesis-node\ (reduce-past\ G\ a)\ b \implies blockDAG.is-genesis-node\ G\ b$ 
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  $blockDAG.is-genesis-node\ (reduce-past\ G\ a)\ b \implies blockDAG.is-genesis-node\ G\ b$ 
    using is-gen by auto
qed

```

```

lemma (in blockDAG) reduce-past-gen-eq:
  assumes  $\neg is-genesis-node\ a$ 
  and  $a \in verts\ G$ 
  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*:
 $e \in \text{arcs } (\text{reduce-past-refl } G \ a) \implies e \in \text{arcs } G$
 using *reduce-past-arcs* by *auto*

lemma (in *blockDAG*) *reduce-past-refl-digraph*:
 assumes $a \in \text{verts } G$
 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 $a \in \text{verts } G$
 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 $\forall e. u \rightarrow^+ \text{pre-digraph.del-arc } (\text{reduce-past-refl } G \ a) \ e \ v$
 $\longrightarrow \neg \text{wf-digraph.arc } (\text{reduce-past-refl } G \ a) \ e \ (u, v)$

proof *safe*

fix *e*

assume *a*: *wf-digraph.arc* (*reduce-past-refl* *G a*) *e* (*u*, *v*)

and *b*: $u \rightarrow^+ \text{pre-digraph.del-arc } (\text{reduce-past-refl } G \ a) \ e \ v$

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 $e \in \text{arcs } (\text{reduce-past-refl } G \ a) \wedge \text{tail } (\text{reduce-past-refl } G \ a) \ e = u$
 $\wedge \text{head } (\text{reduce-past-refl } G \ a) \ e = v$

using *wf-digraph.arcE arc-def a*

by (*metis* (*no-types*))


```

    then show  $\text{arc } e (u, v)$ 
      using  $\text{arc-def reduce-past-refl.simps}$  by auto
  qed
  have  $u \rightarrow^+ \text{pre-digraph.del-arc } G e v$ 
    using  $a b \text{ reduce-past-refl-digraph del-arc-subgraph digraph-axioms}$ 
       $\text{digraphI-induced past-nodes-refl-verts reduce-past-refl.simps}$ 
       $\text{reduce-past-refl-induced-subgraph subgraph-induce-subgraphI arcs-ends-mono}$ 
    trans-mono
    by metis
  then show False
    using  $\text{edge only-new}$  by simp
  qed
next
  obtain  $p$  where  $\text{gen: is-genesis-node } p$  using  $\text{genesis-existAlt}$  by auto
  have  $pe: p \in \text{verts } (\text{reduce-past-refl } G a)$ 
    using  $\text{genesisAlt induce-reachable-preserves-paths}$ 
       $\text{reduce-past.simps past-nodes.simps reachable1-reachable induce-subgraph-verts}$ 
       $\text{gen mem-Collect-eq reachable-neq-reachable1}$ 
    assms by force
  have  $\text{reaches: } (\forall r. r \in \text{verts } (\text{reduce-past-refl } G a) \longrightarrow$ 
     $(r \rightarrow^+ \text{reduce-past-refl } G a p \vee r = p))$ 
  proof safe
    fix  $r$ 
    assume  $\text{in-past: } r \in \text{verts } (\text{reduce-past-refl } G a)$ 
    assume  $\text{une: } r \neq p$ 
    then have  $\text{con: } r \rightarrow^* p$  using  $\text{gen genesisAlt reachable-in-verts}$ 
       $\text{reachable1-reachable}$ 
    by ( $\text{metis in-past induce-subgraph-verts}$ 
       $\text{past-nodes-refl-verts reduce-past-refl.simps subsetD}$ )
    have  $a \rightarrow^* r$  using  $\text{in-past}$  by auto
    then have  $\text{reach: } r \rightarrow^* G \upharpoonright \{w. a \rightarrow^* w\} p$ 
  proof(induction)
    case base
    then show ?case
      using  $\text{con induce-reachable-preserves-paths}$ 
      by ( $\text{metis}$ )
  next
    case ( $\text{step } x y$ )
    then show ?case
  proof –
    have  $\text{Collect } (\text{reachable } G y) \subseteq \text{Collect } (\text{reachable } G x)$ 
      using  $\text{adj-reachable-trans step.hyps(1)}$  by force
    then show ?thesis
      using  $\text{reachable-induce-ss step.IH reachable-neq-reachable1}$ 
      by metis
  qed
  qed
  then show  $r \rightarrow^+ \text{reduce-past-refl } G a p$  unfolding  $\text{reduce-past-refl.simps}$ 
     $\text{past-nodes-refl.simps}$  using  $\text{reachable-in-verts une wf-digraph.reachable-neq-reachable1}$ 

```

```

    by (metis (mono-tags, lifting) Collect-cong wellformed-induce-subgraph)
  qed
  then show  $\exists p \in \text{verts } (\text{reduce-past-refl } G \ a).$  ( $\forall r. r \in \text{verts } (\text{reduce-past-refl } G \ a)$ 
 $\longrightarrow (r \rightarrow^+_{\text{reduce-past-refl } G \ a} p \vee r = p)$ ) 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 $\neg \text{arcs gen-graph} = \{\}$
then have *ex: $\exists a. a \in (\text{arcs gen-graph})$*
by *blast*
also have $\forall a. a \in (\text{arcs gen-graph}) \longrightarrow \text{tail } G \ a = \text{head } G \ a$
proof *safe*
fix *a*
assume $a \in \text{arcs gen-graph}$
then show $\text{tail } G \ a = \text{head } G \ a$
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) ≡ 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) ⇒
      u →+ pre-digraph.del-arc gen-graph e v ⇒ False
      by simp
  next
    have refl: genesis-node →* gen-graph genesis-node
      using gen-gen rtrancl-on-refl
      by (simp add: reachable-def)
    have ∀ r. r ∈ verts gen-graph ⟶ r →* gen-graph genesis-node
  proof safe
    fix r
    assume r ∈ verts gen-graph
    then have r = genesis-node
      using gen-gen by auto
    then show r →* gen-graph genesis-node
      by (simp add: local.refl)
  qed
  then show ∃ p ∈ verts gen-graph.
    (∀ r. r ∈ verts gen-graph ⟶ r →+ gen-graph p ∨ r = p)
    by (simp add: gen-gen)
  qed

lemma (in blockDAG) no-empty-blockDAG:
  shows card (verts G) > 0
proof -
  have ∃ p. p ∈ 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 ⟷ 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:  $\bigwedge V. (blockDAG\ V \implies card\ (verts\ V) = 1 \implies P\ V)$ 
   $\bigwedge W\ c. (\bigwedge V. (blockDAG\ V \implies card\ (verts\ V) = c \implies P\ V))$ 
 $\implies (blockDAG\ W \implies card\ (verts\ W) = Suc\ c \implies P\ W)$ 
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.premis)
  qed
qed

```

lemma *blockDAG-nat-less-induct*[consumes 1, case-names base step]:

```

assumes
  bD: blockDAG Z
and
  cases:  $\bigwedge V. (blockDAG\ V \implies card\ (verts\ V) = 1 \implies P\ V)$ 
   $\bigwedge W\ c. (\bigwedge V. (blockDAG\ V \implies card\ (verts\ V) < c \implies P\ V))$ 
 $\implies (blockDAG\ W \implies card\ (verts\ W) = c \implies P\ W)$ 
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:
       $(\bigwedge Za. card\ (verts\ Za) < card\ (verts\ Z) \implies blockDAG\ Za \implies 0 < card\ (verts\ Za) \implies P\ Za)$ 
    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  $G \in$  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  $G \neq \{\}$ 
  then obtain  $z$  where part-of:  $z \in$  arcs  $G$ 
    by auto
  then have tail: tail  $G$   $z \in$  verts  $G$ 
    using wf-digraph-def blockDAG-def DAG-def
      digraph-def blockDAG-axioms nomulti-digraph.axioms(1)
    by metis
  also have head: head  $G$   $z \in$  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  $\neg$  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  $\longleftrightarrow$  ( $\exists b$   $H$ . (blockDAG  $H \wedge b \in$  verts  $G \wedge$  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}) ≠ {} using card-gt-0-iff by blast
  then obtain y where y-def: y ∈ (verts G) - {x} by auto
  then have uneq: y ≠ x by auto
  have y-in: y ∈ (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 ∈ (verts G) - {x} ∧ is-tip G z using tips-exist
    is-tip.simps by auto
  then have uneq: z ≠ x by auto
  have z-in: z ∈ 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 (∃ b H. blockDAG H ∧ b ∈ verts G ∧ 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 ∈ 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) (∃ b H. (blockDAG H ∧ b ∈ verts G ∧ del-vert b = H))
  using blockDAG-cases-one blockDAG-cases-more
    blockDAG-size-cases by auto

```

```

lemma (in blockDAG) blockDAG-cases-more2:
  assumes card (verts G) > 1
  shows ( $\exists b H. (blockDAG\ H \wedge b \in \text{verts } G \wedge \text{del-vert } b = H \wedge (\forall c \in \text{verts } G. \neg c \rightarrow^+_G b))$ )
proof –
  obtain tip where tip-tip: is-tip G tip using tips-exist by auto
  then have tip-neq-gen:  $\neg \text{is-genesis-node } tip$ 
    using assms tips-unequal-gen by auto
  have tip-in: tip  $\in \text{verts } G$  using tip-tip is-tip.simps by metis
  have nre: ( $\forall c. \neg c \rightarrow^+_G tip$ ) using tips-not-referenced tip-tip by auto
  let ?H = del-vert tip
  have blockDAG ?H using del-tips-bDAG tip-tip tip-neq-gen by auto
  then show ?thesis using nre tip-in
    by blast
qed

```

```

lemma (in blockDAG) blockDAG-cases2:
  obtains (base) (G = gen-graph)
  | (more) ( $\exists b H. (blockDAG\ H \wedge b \in \text{verts } G \wedge \text{del-vert } b = H \wedge (\forall c \in \text{verts } G. \neg c \rightarrow^+_G b))$ )
  using blockDAG-cases-one blockDAG-cases-more2
    blockDAG-size-cases by auto

```

```

lemma blockDAG-induct[consumes 1, case-names base step]:
  assumes fund: blockDAG G
  assumes cases:  $\bigwedge V::('a,'b) \text{ pre-digraph. } blockDAG\ V \Longrightarrow P (blockDAG.gen-graph\ V)$ 
     $\bigwedge H::('a,'b) \text{ pre-digraph. } (\bigwedge b::'a. blockDAG (pre-digraph.del-vert\ H\ b) \Longrightarrow b \in \text{verts } H \Longrightarrow P (pre-digraph.del-vert\ H\ b))$ 
     $\Longrightarrow (blockDAG\ H \Longrightarrow P\ H)$ 
  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 ( $\bigwedge V. blockDAG\ V \Longrightarrow \text{card } (\text{verts } V) = c \Longrightarrow P\ V$ )  $\Longrightarrow$ 
     $blockDAG\ W \Longrightarrow \text{card } (\text{verts } W) = \text{Suc } c \Longrightarrow P\ W$ 
proof –

```

```

assume ind:  $\bigwedge V. (blockDAG\ V \implies card\ (verts\ V) = c \implies P\ V)$ 
and bD: blockDAG W
and size:  $card\ (verts\ W) = Suc\ c$ 
have assm2:  $\bigwedge b. blockDAG\ (pre-digraph.del-vert\ W\ b)$ 
 $\implies b \in verts\ W \implies P(pre-digraph.del-vert\ W\ b)$ 
proof –
  fix b
  assume bD2: blockDAG (pre-digraph.del-vert W b)
  assume in-verts:  $b \in verts\ W$ 
  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  $\Rightarrow$  'a
  where genesis-nodeAlt G = (if ( $\neg blockDAG\ G$ ) 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 ( $\lambda G. (card\ (verts\ G))$ )
  show wf ?R by auto
next
  fix G :: ('a::linorder,'b) pre-digraph
  assume  $\neg \neg blockDAG\ G$ 
  then have bD: blockDAG G by simp
  assume  $card\ (verts\ G) \neq 1$ 
  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  $hd\ (sorted-list-of-set\ (tips\ G)) \in tips\ G$ 
    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)  $\in measure\ (\lambda G. card\ (verts\ G))$ 
    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 –

interpret *B*: *blockDAG* *G* **using** *assms*(1) **by** *simp*

have *exone*: $\exists! x. x \in (\text{verts } G)$

using *one* *B.genesis-in-verts* *B.genesis-unique-exists* *B.reduce-less*

B.reduce-past-dagbased less-nat-zero-code less-one *B.gen-gen* *B.gen-graph-all-one*

singleton-iff

by (*metis*)

then have *sorted-list-of-set* (*verts* *G*) $\neq \square$

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* \in *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*: *B.is-genesis-node* (*genesis-nodeAlt* *G*)

using *one* *B.blockDAG-size-cases* *B.reduce-less*

B.reduce-past-dagbased less-one *B.genesis-unique-exists* *B.is-genesis-node.elims*(2)

exone

by (*metis*)

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*:

($\bigwedge V::('a,'b) \text{ pre-digraph. } \text{blockDAG } V \implies \text{card } (\text{verts } V) < c \implies$

blockDAG.is-genesis-node *V* (*genesis-nodeAlt* *V*))

assume *bD*: *blockDAG* *W*

interpret *B*: *blockDAG* *W* **using** *bD* **by** *simp*

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:  $1 \neq \text{card } (\text{verts } W)$  by auto
      have se: set (sorted-list-of-set (tips W)) = tips W
        by (simp add: tips-def)
      obtain a where a-def:  $a = \text{hd } (\text{sorted-list-of-set } (\text{tips } W))$ 
        by simp
      have tip:  $a \in \text{tips } W$ 
        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:  $a \in \text{verts } W$ 
        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 is-tip W a using tip CollectD unfolding tips-def by simp
      then have n-gen:  $\neg B.\text{is-genesis-node } a$ 
        using B.tips-unequal-gen more by simp
      then have bD2: blockDAG (reduce-past W a)
        using B.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  $\Rightarrow$  'a  $\Rightarrow$  int  $\Rightarrow$  int
  where tie-break-int a b i =
    (if i=0 then (if (b < a) then -1 else 1) else i)
```

Sign function with 0

```
fun signum :: int  $\Rightarrow$  int
  where signum a = (if a > 0 then 1 else if a < 0 then -1 else 0)
```

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  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  'a  $\Rightarrow$  int
  where
    vote-Spectre V a b c = (
      if ( $\neg$  blockDAG V  $\vee$  a  $\notin$  verts V  $\vee$  b  $\notin$  verts V  $\vee$  c  $\notin$  verts V) then 0 else
      if (b=c) then 1 else
      if (((a  $\rightarrow^+$  V b)  $\vee$  a = b)  $\wedge$   $\neg$ (a  $\rightarrow^+$  V c)) then 1 else
      if (((a  $\rightarrow^+$  V c)  $\vee$  a = c)  $\wedge$   $\neg$ (a  $\rightarrow^+$  V b)) then -1 else
      if ((a  $\rightarrow^+$  V b)  $\wedge$  (a  $\rightarrow^+$  V c)) then
        (tie-break-int b c (signum (sum-list (map ( $\lambda$ i.
          (vote-Spectre (reduce-past V a) i b c)) (sorted-list-of-set (past-nodes V a))))))
        else
        signum (sum-list (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

[λ (V, a, b, c). card (verts V),

λ (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:  $\neg (\neg \text{blockDAG } V \vee a \notin \text{verts } V \vee b \notin \text{verts } V \vee c \notin \text{verts } V)$ 
then have a-in:  $a \in \text{verts } V$  using bD by simp
assume x  $\in \text{set } (\text{sorted-list-of-set } (\text{future-nodes } V \ a))$ 
then have x  $\in \text{future-nodes } V \ a$  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  $\forall x. \{e. e \rightarrow^* V \ x\} \subseteq \text{verts } V$  using subs bD2 subsetI
      wf-digraph.reachable-in-verts(1) mem-Collect-eq
by metis
then have fin:  $\forall x. \text{finite } \{e. e \rightarrow^* V \ x\}$  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  $\text{card } \{e. e \rightarrow^* V \ x\} < \text{card } \{e. e \rightarrow^* V \ a\}$  using fin
by (simp add: psubset-card-mono)
then show  $((V, x, b, c), V, a, b, c)$ 
       $\in \text{measures}$ 
       $[\lambda(V, a, b, c). \text{card } (\text{verts } V), \lambda(V, a, b, c). \text{card } \{e. e \rightarrow^* V \ a\}]$ 
by simp
qed

```

Given vote-Spectre calculate if $a < b$ for arbitrary nodes

```

definition Spectre-Order :: ('a::linorder, 'b) pre-digraph  $\Rightarrow 'a \Rightarrow 'a \Rightarrow \text{bool}$ 
  where Spectre-Order G a b = ( tie-break-int a b (signum ( sum-list (map ( $\lambda i.$ 
    (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 :: ('a::linorder, 'b) pre-digraph  $\Rightarrow ('a \times 'a) \text{ set}$ 
  where SPECTRE G  $\equiv \{(a, b) \in (\text{verts } G \times \text{verts } G). \text{Spectre-Order } G \ a \ b\}$ 

```

4.2 Lemmas

```

lemma sumlist-one-mono:
  assumes  $\forall x \in \text{set } L. x \geq 0$ 
  and  $\exists x \in \text{set } L. x > 0$ 
  shows  $\text{signum } (\text{sum-list } L) = 1$ 
  using assms

```

```

proof(induct L, simp)
  case (Cons a2 L)
  consider (bg)  $a2 > 0 \mid a2 = 0$  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 tie-break-int.simps
      by auto
  next
  case 2
  then have be:  $\exists a \in \text{set } L. 0 < a$  using Cons
    by (metis less-int-code(1) set-ConsD)
  then have  $L \neq []$  by auto
  then show ?thesis using sum-list-def 2
    using Cons.hyps Cons.prem(1) be by auto
qed
qed

```

lemma *domain-signum*: *signum i* $\in \{-1, 0, 1\}$ **by** *simp*

lemma *signum-mono*:
assumes $i \leq j$
shows *signum i* \leq *signum j* **using** *assms* **by** *simp*

lemma *tie-break-mono*:
assumes $i \leq j$
shows *tie-break-int b c i* \leq *tie-break-int b c j* **using** *assms* **by** *simp*

lemma *domain-tie-break*:
shows *tie-break-int a b (signum i)* $\in \{-1, 1\}$
using *tie-break-int.simps*
by *auto*

lemma *Spectre-casesAlt*:
fixes $V :: ('a::\text{linorder}, 'b) \text{ pre-digraph}$
and $a :: 'a::\text{linorder}$ **and** $b :: 'a::\text{linorder}$ **and** $c :: 'a::\text{linorder}$
obtains (*no-bD*) $(\neg \text{blockDAG } V \vee a \notin \text{verts } V \vee b \notin \text{verts } V \vee c \notin \text{verts } V)$
 \mid (*equal*) $(\text{blockDAG } V \wedge a \in \text{verts } V \wedge b \in \text{verts } V \wedge c \in \text{verts } V) \wedge b = c$
 \mid (*one*) $(\text{blockDAG } V \wedge a \in \text{verts } V \wedge b \in \text{verts } V \wedge c \in \text{verts } V) \wedge$
 $b \neq c \wedge (((a \rightarrow^+_V b) \vee a = b) \wedge \neg(a \rightarrow^+_V c))$
 \mid (*two*) $(\text{blockDAG } V \wedge a \in \text{verts } V \wedge b \in \text{verts } V \wedge c \in \text{verts } V) \wedge b \neq c$
 $\wedge \neg(((a \rightarrow^+_V b) \vee a = b) \wedge \neg(a \rightarrow^+_V c)) \wedge$

$((a \rightarrow^+_V c) \vee a = c) \wedge \neg(a \rightarrow^+_V b)$
 $|$ (three) ($blockDAG\ V \wedge a \in \text{verts}\ V \wedge b \in \text{verts}\ V \wedge c \in \text{verts}\ V) \wedge b \neq c$
 $\wedge \neg(((a \rightarrow^+_V b) \vee a = b) \wedge \neg(a \rightarrow^+_V c)) \wedge$
 $\neg(((a \rightarrow^+_V c) \vee a = c) \wedge \neg(a \rightarrow^+_V b)) \wedge$
 $((a \rightarrow^+_V b) \wedge (a \rightarrow^+_V c))$
 $|$ (four) ($blockDAG\ V \wedge a \in \text{verts}\ V \wedge b \in \text{verts}\ V \wedge c \in \text{verts}\ V) \wedge b \neq c \wedge$
 $\neg(((a \rightarrow^+_V b) \vee a = b) \wedge \neg(a \rightarrow^+_V c)) \wedge$
 $\neg(((a \rightarrow^+_V c) \vee a = c) \wedge \neg(a \rightarrow^+_V b)) \wedge$
 $\neg((a \rightarrow^+_V b) \wedge (a \rightarrow^+_V c))$
by *auto*

lemma *domain-Spectre*:

shows *vote-Spectre* $V\ a\ b\ c \in \{-1, 0, 1\}$

proof(*rule vote-Spectre.cases, auto*) **qed**

lemma *antisymmetric-tie-break*:

shows $b \neq c \implies \text{tie-break-int}\ b\ c\ i = -\ \text{tie-break-int}\ c\ b\ (-i)$

unfolding *tie-break-int.simps* **using** *less-not-sym* **by** *auto*

lemma *antisymmetric-sumlist*:

shows $\text{sum-list}\ (l :: \text{int list}) = -\ \text{sum-list}\ (\text{map}\ (\lambda x. -x)\ l)$

proof(*induct l, auto*) **qed**

lemma *antisymmetric-signum*:

shows $\text{signum}\ i = -\ (\text{signum}\ (-i))$

by *auto*

lemma *append-diff-sorted-set*:

assumes $a \in A$

and *finite* A

shows $\text{sum-list}\ ((\text{map}\ (P :: 'a :: \text{linorder} \Rightarrow \text{int})))$

$(\text{sorted-list-of-set}\ (A - \{a\}))$

$= \text{sum-list}\ ((\text{map}\ P)(\text{sorted-list-of-set}\ (A))) - (P\ a)$

proof –

let $?L1 = (\text{sorted-list-of-set}\ (A))$

have *d-1*: $\text{distinct}\ ?L1$ **using** *sum-list-distinct-conv-sum-set sorted-list-of-set(2)*

by *auto*

then have *s-1*: $\text{sum-list}\ ((\text{map}\ P)\ ?L1)$

$= \text{sum}\ P\ (\text{set}\ ?L1)$ **using** *sum-list-distinct-conv-sum-set* **by** *metis*

let $?L2 = (\text{sorted-list-of-set}\ (A - \{a\}))$

have *d-2*: $\text{distinct}\ ?L2$ **using** *sum-list-distinct-conv-sum-set sorted-list-of-set(2)*

by *auto*

```

then have s-2: sum-list ((map P) ?L2)
= sum P (set ?L2) using sum-list-distinct-conv-sum-set by metis
have s-3: sum P (set ?L2) = sum P (set ?L1) - (P a)
using assms sorted-list-of-set(1)
by (simp add: sum-diff1)
show ?thesis
unfolding s-1 s-2 s-3 by simp
qed

```

```

lemma append-diff-sorted-set2:
assumes a ∈ A
and b ∈ A
and a ≠ b
and finite A
shows sum-list ((map (P::('a::linorder ⇒ int)))
(sorted-list-of-set (A - {a} - {b}))))
= sum-list ((map P)(sorted-list-of-set (A))) - (P a) - (P b)
using assms append-diff-sorted-set
by (metis finite-Diff insert-Diff insert-iff)

```

```

lemma vote-Spectre-one-exists:
assumes blockDAG V
and a ∈ verts V
and b ∈ verts V
shows ∃ i ∈ verts V. vote-Spectre V i a b ≠ 0
proof
show a ∈ verts V using assms(2) by simp
show vote-Spectre V a a b ≠ 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
end

```

```

theory Ghostdag
imports blockDAG Utils TopSort
begin

```

5 GHOSTDAG

Based on the GHOSTDAG blockDAG consensus algorithmus by Sompolinsky and Zohar 2018

5.1 Funcitions and Definitions

Function to compare the size of set and break ties. Used for the GHOSTDAG maximum blue cluster selection

```
fun larger-blue-tuple ::
  (('a::linorder set × 'a list) × 'a) ⇒ (('a set × 'a list) × 'a) ⇒ (('a set × 'a list)
  × 'a)
  where larger-blue-tuple A B =
    (if (card (fst (fst A))) > (card (fst (fst B)))) ∨
    (card (fst (fst A)) ≥ card (fst (fst B)) ∧ snd A ≤ snd B) then A else B
```

Function to add node a to a tuple of a set S and List L

```
fun add-set-list-tuple :: (('a::linorder set × 'a list) × 'a) ⇒ ('a::linorder set × 'a
  list)
  where add-set-list-tuple ((S,L),a) = (S ∪ {a}, L @ [a])
```

Function that adds a node a to a kCluster S , if $S + a$ remains a kCluster. Also adds a to the end of list L

```
fun app-if-blue-else-add-end ::
  ('a::linorder, 'b) pre-digraph ⇒ nat ⇒ 'a ⇒ ('a::linorder set × 'a list)
  ⇒ ('a::linorder set × 'a list)
  where app-if-blue-else-add-end G k a (S,L) = (if (kCluster G k (S ∪ {a})))
  then add-set-list-tuple ((S,L),a) else (S,L @ [a])
```

Function to select the largest $((S, L), a)$ according to *larger – blue – tuple*

```
fun choose-max-blue-set :: (('a::linorder set × 'a list) × 'a) list ⇒ (('a set × 'a
  list) × 'a)
  where choose-max-blue-set L = fold (larger-blue-tuple) L (hd L)
```

GHOSTDAG ordering algorithm

```
function OrderDAG :: ('a::linorder, 'b) pre-digraph ⇒ nat ⇒ ('a set × 'a list)
  where
    OrderDAG G k =
      (if (¬ blockDAG G) then ({},[]) else
       if (card (verts G) = 1) then ({genesis-nodeAlt G},[genesis-nodeAlt G]) else
       let M = choose-max-blue-set
         ((map (λi.(((OrderDAG (reduce-past G i) k) , i)) (sorted-list-of-set (tips G))))
          in fold (app-if-blue-else-add-end G k) (top-sort G (sorted-list-of-set (anticone G
            (snd M))))
          (add-set-list-tuple M))
    by auto
```



```

termination proof
  let ?R = measure (  $\lambda(G, k). (card (verts G))$  )
  show wf ?R by auto
next
  fix G::('a::linorder,'b) pre-digraph
  fix k::nat
  fix x
  assume bD:  $\neg \neg blockDAG\ G$ 
  assume card (verts G)  $\neq 1$ 
  then have card (verts G) > 1 using bD blockDAG.blockDAG-size-cases by auto

  then have nT:  $\forall x \in tips\ G. \neg blockDAG.is-genesis-node\ G\ x$ 
    using blockDAG.tips-unequal-gen bD tips-def mem-Collect-eq
    by metis
  assume x  $\in set\ (sorted-list-of-set\ (tips\ G))$ 
  then have in-t:  $x \in tips\ G$  using bD
  by (metis card-gt-0-iff length-pos-if-in-set length-sorted-list-of-set set-sorted-list-of-set)

  then show ((reduce-past G x, k), G, k)  $\in measure\ (\lambda(G, k). card\ (verts\ G))$ 
    using blockDAG.reduce-less bD tips-def is-tip.simps
    by fastforce
qed

```

Creating a relation on $verts\ G$ based on the GHOSTDAG OrderDAG algorithm

```

fun GHOSTDAG :: nat  $\Rightarrow$  ('a::linorder,'b) pre-digraph  $\Rightarrow$  'a rel
  where GHOSTDAG k G = list-to-rel (snd (OrderDAG G k))

```

5.2 Soundness

```

lemma OrderDAG-casesAlt:
  obtains (ntB)  $\neg blockDAG\ G$ 
  | (one)  $blockDAG\ G \wedge card\ (verts\ G) = 1$ 
  | (more)  $blockDAG\ G \wedge card\ (verts\ G) > 1$ 
  using blockDAG.blockDAG-size-cases by auto

```

5.2.1 Soundness of the add – set – list function

```

lemma add-set-list-tuple-mono:
  shows set L  $\subseteq set\ (snd\ (add-set-list-tuple\ ((S,L),a)))$ 
  using add-set-list-tuple.simps by auto

```

```

lemma add-set-list-tuple-mono2:
  shows set (snd (add-set-list-tuple ((S,L),a)))  $\subseteq set\ L \cup \{a\}$ 
  using add-set-list-tuple.simps by auto

```

```

lemma add-set-list-tuple-length:
  shows length (snd (add-set-list-tuple ((S,L),a))) = Suc (length L)
proof(induct L, auto) qed

```

5.2.2 Soundness of the *add – if – blue* function

lemma *app-if-blue-mono*:

assumes *finite S*

shows $(fst (S, L)) \subseteq (fst (app-if-blue-else-add-end G k a (S, L)))$

unfolding *app-if-blue-else-add-end.simps add-set-list-tuple.simps*

by (*simp add: assms card-mono subset-insertI*)

lemma *app-if-blue-mono2*:

shows $set (snd (S, L)) \subseteq set (snd (app-if-blue-else-add-end G k a (S, L)))$

unfolding *app-if-blue-else-add-end.simps add-set-list-tuple.simps*

by (*simp add: subsetI*)

lemma *app-if-blue-append*:

shows $a \in set (snd (app-if-blue-else-add-end G k a (S, L)))$

unfolding *app-if-blue-else-add-end.simps add-set-list-tuple.simps*

by *simp*

lemma *app-if-blue-mono3*:

shows $set (snd (app-if-blue-else-add-end G k a (S, L))) \subseteq set L \cup \{a\}$

unfolding *app-if-blue-else-add-end.simps add-set-list-tuple.simps*

by (*simp add: subsetI*)

lemma *app-if-blue-mono4*:

assumes $set L1 \subseteq set L2$

shows $set (snd (app-if-blue-else-add-end G k a (S, L1)))$

$\subseteq set (snd (app-if-blue-else-add-end G k a (S2, L2)))$

unfolding *app-if-blue-else-add-end.simps add-set-list-tuple.simps*

using *assms by auto*

lemma *app-if-blue-card-mono*:

assumes *finite S*

shows $card (fst (S, L)) \leq card (fst (app-if-blue-else-add-end G k a (S, L)))$

unfolding *app-if-blue-else-add-end.simps add-set-list-tuple.simps*

by (*simp add: assms card-mono subset-insertI*)

lemma *app-if-blue-else-add-end-length*:

shows $length (snd (app-if-blue-else-add-end G k a (S, L))) = Suc (length L)$

proof(*induction L, auto*) **qed**

5.2.3 Soundness of the *larger – blue – tuple* comparison

lemma *larger-blue-tuple-mono*:

assumes *finite (fst V)*

shows $larger-blue-tuple ((app-if-blue-else-add-end G k a V), b) (V, b)$

$= ((app-if-blue-else-add-end G k a V), b)$

using *assms app-if-blue-card-mono larger-blue-tuple.simps eq-refl*
by (*metis fst-conv prod.collapse snd-conv*)

lemma *larger-blue-tuple-subs*:
shows *larger-blue-tuple* $A\ B \in \{A,B\}$ **by** *auto*

5.2.4 Soundness of the *choose_max_blue_set* function

lemma *choose-max-blue-avoid-empty*:
assumes $L \neq []$
shows *choose-max-blue-set* $L \in \text{set } L$
unfolding *choose-max-blue-set.simps*
proof (*rule fold-invariant*)
show $\bigwedge x. x \in \text{set } L \implies x \in \text{set } L$ **using** *assms* **by** *auto*
next
show $\text{hd } L \in \text{set } L$ **using** *assms* **by** *auto*
next
fix $x\ s$
assume $x \in \text{set } L$
and $s \in \text{set } L$
then show *larger-blue-tuple* $x\ s \in \text{set } L$ **using** *larger-blue-tuple.simps* **by** *auto*
qed

5.2.5 Auxiliary lemmas for OrderDAG

lemma *fold-app-length*:
shows $\text{length } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G\ k)\ L1\ PL2)) = \text{length } L1 + \text{length } (\text{snd } PL2)$
proof(*induct L1 arbitrary: PL2*)
case *Nil*
then show *?case* **by** *auto*
next
case (*Cons a L1*)
then show *?case* **unfolding** *fold-Cons comp-apply* **using** *app-if-blue-else-add-end-length*
by (*metis add-Suc add-Suc-right length-Cons old.prod.exhaust snd-conv*)
qed

lemma *fold-app-mono*:
shows $\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G\ k)\ L2\ (S, L1)) = L1 @ L2$
proof(*induct L2 arbitrary: S L1, simp*)
case (*Cons a L2*)
then show *?case* **unfolding** *fold-simps(2)* **using** *app-if-blue-else-add-end.simps*
by *simp*
qed

lemma *fold-app-mono1*:
assumes $x \in \text{set } (\text{snd } (S, L1))$
shows $x \in \text{set } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G\ k)\ L2\ (S2, L1)))$

```

using fold-app-mono
by (metis Cons-eq-appendI append.assoc assms in-set-conv-decomp sndI)

lemma fold-app-mono2:
  assumes  $x \in \text{set } L2$ 
  shows  $x \in \text{set } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G \ k) \ L2 \ (S, L1)))$ 
  using assms unfolding fold-app-mono by auto

lemma fold-app-mono3:
  assumes  $\text{set } L1 \subseteq \text{set } L2$ 
  shows  $\text{set } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G \ k) \ L \ (S1, L1)))$ 
     $\subseteq \text{set } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G \ k) \ L \ (S2, L2)))$ 
  using assms unfolding fold-app-mono
  by auto

lemma fold-app-mono-ex:
  shows  $\text{set } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G \ k) \ L2 \ (S, L1))) = (\text{set } L2 \cup \text{set } L1)$ 
  unfolding fold-app-mono by auto

lemma fold-app-mono-rel:
  assumes  $(x, y) \in \text{list-to-rel } L1$ 
  shows  $(x, y) \in \text{list-to-rel } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G \ k) \ L2 \ (S, L1)))$ 
  using assms
proof(induct L2 arbitrary: S L1, simp)
  case (Cons a L2)
  then show ?case
    unfolding fold.simps(2) comp-apply
    using list-to-rel-mono app-if-blue-else-add-end.simps
    by (metis add-set-list-tuple.simps prod.collapse snd-conv)
qed

lemma fold-app-mono-rel2:
  assumes  $(x, y) \in \text{list-to-rel } L2$ 
  shows  $(x, y) \in \text{list-to-rel } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G \ k) \ L2 \ (S, L1)))$ 
  using assms
  by (simp add: fold-app-mono list-to-rel-mono2)

lemma fold-app-app-rel:
  assumes  $x \in \text{set } L1$ 
  and  $y \in \text{set } L2$ 
  shows  $(x, y) \in \text{list-to-rel } (\text{snd } (\text{fold } (\text{app-if-blue-else-add-end } G \ k) \ L2 \ (S, L1)))$ 
  using assms
proof(induct L2 arbitrary: S L1, simp)
  case (Cons a L2)
  then show ?case
    unfolding fold.simps(2) comp-apply

```

```

    using list-to-rel-append app-if-blue-else-add-end.simps
  by (metis Un-iff add-set-list-tuple.simps fold-app-mono-rel set-ConsD set-append)

qed

lemma chosen-max-tip:
  assumes blockDAG G
  assumes  $x = \text{snd } (\text{choose-max-blue-set } (\text{map } (\lambda i. (\text{OrderDAG } (\text{reduce-past } G \ i) \ k, \ i))$ 
     $(\text{sorted-list-of-set } (\text{tips } G))))$ 
  shows  $x \in \text{set } (\text{sorted-list-of-set } (\text{tips } G))$  and  $x \in \text{tips } G$ 
proof -
  obtain pp where pp-in:  $pp = (\text{map } (\lambda i. (\text{OrderDAG } (\text{reduce-past } G \ i) \ k, \ i))$ 
     $(\text{sorted-list-of-set } (\text{tips } G)))$  using blockDAG.tips-exist by auto
  have mm:  $\text{choose-max-blue-set } pp \in \text{set } pp$  using pp-in choose-max-blue-avoid-empty
    digraph.tips-finite subs assms(1)
    list.map-disc-iff sorted-list-of-set-eq-Nil-iff blockDAG.tips-not-empty
    by (metis (mono-tags, lifting))
  then have kk:  $\text{snd } (\text{choose-max-blue-set } pp) \in \text{set } (\text{map } \text{snd } pp)$ 
    by auto
  have mm2:  $\bigwedge L. (\text{map } \text{snd } (\text{map } (\lambda i. ((\text{OrderDAG } (\text{reduce-past } G \ i) \ k), \ i)) \ L)) \ L)$ 
  = L
  proof -
    fix L
    show  $\text{map } \text{snd } (\text{map } (\lambda i. (\text{OrderDAG } (\text{reduce-past } G \ i) \ k, \ i)) \ L) = L$ 
    proof(induct L)
      case Nil
      then show ?case by auto
    next
      case (Cons a L)
      then show ?case by auto
    qed
  qed
  have  $\text{set } (\text{map } \text{snd } pp) = \text{set } (\text{sorted-list-of-set } (\text{tips } G))$ 
    using mm2 pp-in by auto
  then show  $x \in \text{set } (\text{sorted-list-of-set } (\text{tips } G))$  using pp-in assms(2) kk by blast

  then show  $x \in \text{tips } G$ 
    using digraph.tips-finite sorted-list-of-set(1) kk subs assms pp-in by auto
qed

```

```

lemma chosen-map-simps1:
  assumes  $x \in \text{set } (\text{map } (\lambda i. (P \ i, \ i)) \ L)$ 
  shows  $\text{fst } x = P (\text{snd } x)$ 
  using assms
proof(induct L, auto) qed

```

```

lemma chosen-map-simps:

```

```

assumes blockDAG G
assumes x = map ( $\lambda i.$  (OrderDAG (reduce-past G i) k, i))
  (sorted-list-of-set (tips G))
shows snd (choose-max-blue-set x)  $\in$  set (sorted-list-of-set (tips G))
  and snd (choose-max-blue-set x)  $\in$  tips G
  and set (map snd x) = set (sorted-list-of-set (tips G))
  and choose-max-blue-set x  $\in$  set x
  and  $\neg$  blockDAG.is-genesis-node G (snd (choose-max-blue-set x))  $\implies$ 
    blockDAG (reduce-past G (snd (choose-max-blue-set x)))
  and OrderDAG (reduce-past G (snd (choose-max-blue-set x))) k = fst (choose-max-blue-set
x)
proof –
  obtain pp where pp-in: pp = (map ( $\lambda i.$  (OrderDAG (reduce-past G i) k, i))
    (sorted-list-of-set (tips G))) using blockDAG.tips-exist by auto
  have mm: choose-max-blue-set pp  $\in$  set pp using pp-in choose-max-blue-avoid-empty
    digraph.tips-finite subs assms(1)
    list.map-disc-iff sorted-list-of-set-eq-Nil-iff blockDAG.tips-not-empty
  by (metis (mono-tags, lifting))
  then have kk: snd (choose-max-blue-set pp)  $\in$  set (map snd pp)
  by auto
  have seteq: set (map snd pp) = set (sorted-list-of-set (tips G))
  using map-snd-map pp-in by auto
  then show snd (choose-max-blue-set x)  $\in$  set (sorted-list-of-set (tips G))
  using pp-in assms(2) kk by blast
  then show tip: snd (choose-max-blue-set x)  $\in$  tips G
  using digraph.tips-finite sorted-list-of-set(1) kk subs assms pp-in by auto
  show set (map snd x) = set (sorted-list-of-set (tips G))
  using map-snd-map assms(2)
  by simp
  then show choose-max-blue-set x  $\in$  set x using seteq pp-in assms(2)
  mm by blast
  show OrderDAG (reduce-past G (snd (choose-max-blue-set x))) k = fst (choose-max-blue-set
x)
  by (metis (no-types) assms(2) chosen-map-simps1 mm pp-in)
  assume  $\neg$  blockDAG.is-genesis-node G (snd (choose-max-blue-set x))
  then show blockDAG (reduce-past G (snd (choose-max-blue-set x)))
  using tip blockDAG.reduce-past-dagbased assms(1) digraph.tips-in-verts subs
subsetD
  by metis
qed

```

5.2.6 OrderDAG soundness

lemma Verts-in-OrderDAG:

```

assumes blockDAG G
  and x  $\in$  verts G
shows x  $\in$  set (snd (OrderDAG G k))
using assms
proof(induct G k arbitrary: x rule: OrderDAG.induct)

```

```

case (1 G k x)
then have bD: blockDAG G by auto
assume x-in: x ∈ verts G
then consider (cD1) card (verts G) = 1 | (cDm) card (verts G) ≠ 1 by auto
then show x ∈ set (snd (OrderDAG G k))
proof(cases)
  case (cD1)
  then have set (snd (OrderDAG G k)) = {genesis-nodeAlt G}
  using 1 OrderDAG.simps by auto
  then show ?thesis using x-in bD cD1
  genesis-nodeAlt-sound blockDAG.is-genesis-node.simps
  using 1
  by (metis card-1-singletonE singletonD)
next
case (cDm)
then show ?thesis
proof –
  obtain pp where pp-in: pp = (map (λi. (OrderDAG (reduce-past G i) k, i))
    (sorted-list-of-set (tips G))) using blockDAG.tips-exist by auto
  then have tt2: snd (choose-max-blue-set pp) ∈ tips G
  using chosen-map-simps bD
  by blast
  show ?thesis
  proof(rule blockDAG.tips-cases)
    show blockDAG G using bD by auto
    show snd (choose-max-blue-set pp) ∈ tips G using tt2 by auto
    show x ∈ verts G using x-in by auto
  next
    assume as1: x = snd (choose-max-blue-set pp)
    obtain fCur where fcur-in: fCur = add-set-list-tuple (choose-max-blue-set
pp)
    by auto
    have x ∈ set (snd(fCur))
    unfolding as1 using add-set-list-tuple.simps fcur-in
    add-set-list-tuple.cases snd-conv insertI1 snd-conv
    by (metis (mono-tags, hide-lams) Un-insert-right fst-conv list.simps(15)
set-append)
    then have x ∈ set (snd (fold (app-if-blue-else-add-end G k)
      (top-sort G (sorted-list-of-set (anticone G (snd (choose-max-blue-set
pp)))))) (fCur)))
    using fold-app-mono1 surj-pair
    by (metis)
    then show ?thesis unfolding pp-in fcur-in using 1 OrderDAG.simps cDm
    by (metis (mono-tags, lifting))
  next
    assume anti: x ∈ anticone G (snd (choose-max-blue-set pp))
    obtain ttt where ttt-in: ttt = add-set-list-tuple (choose-max-blue-set pp) by
auto
    have x ∈ set (snd (fold (app-if-blue-else-add-end G k)

```

```

      (top-sort G (sorted-list-of-set (anticone G (snd (choose-max-blue-set
pp))))))
      tt2))
    using pp-in sorted-list-of-set(1) anti bD subs
    DAG.anticone-finite fold-app-mono2 surj-pair top-sort-con by metis
    then show x ∈ set (snd (OrderDAG G k)) using OrderDAG.simps pp-in
bD cDm tt-in 1
    by (metis (no-types, lifting) map-eq-conv)
  next
    assume as2: x ∈ past-nodes G (snd (choose-max-blue-set pp))
    then have pas: x ∈ verts (reduce-past G (snd (choose-max-blue-set pp)))
    using reduce-past.simps induce-subgraph-verts by auto
    have cd1: card (verts G) > 1 using cDm bD
    using blockDAG.blockDAG-size-cases by blast
    have (snd (choose-max-blue-set pp)) ∈ set (sorted-list-of-set (tips G)) using
tt2
    digraph.tips-finite bD subs sorted-list-of-set(1) by auto
  moreover
    have blockDAG (reduce-past G (snd (choose-max-blue-set pp))) using
    blockDAG.reduce-past-dagbased bD tt2 blockDAG.tips-unequal-gen
    cd1 tips-def CollectD by metis
  ultimately have bass:
    x ∈ set ((snd (OrderDAG (reduce-past G (snd (choose-max-blue-set pp)))
k)))
    using pp-in 1 cDm tt2 pas by metis
  then have in-F: x ∈ set (snd (fst ((choose-max-blue-set pp))))
    using x-in-chosen-map-simps(6) pp-in
    using bD by fastforce
  then have x ∈ set (snd (fold (app-if-blue-else-add-end G k)
(top-sort G (sorted-list-of-set (anticone G (snd (choose-max-blue-set pp))))))
(fst((choose-max-blue-set pp))))))
    by (metis fold-app-mono1 in-F prod.collapse)
  moreover have OrderDAG G k = (fold (app-if-blue-else-add-end G k)
(top-sort G (sorted-list-of-set (anticone G (snd (choose-max-blue-set pp))))))
(add-set-list-tuple (choose-max-blue-set pp))) using cDm 1 OrderDAG.simps
pp-in
    by (metis (no-types, lifting) map-eq-conv)
  then show x ∈ set (snd (OrderDAG G k))
    by (metis (no-types, lifting) add-set-list-tuple-mono fold-app-mono1
in-F prod.collapse subset-code(1))
  qed
qed
qed
qed

```

lemma *OrderDAG-in-verts:*
assumes $x \in \text{set } (\text{snd } (\text{OrderDAG } G \ k))$
shows $x \in \text{verts } G$


```

using assms
proof(induction G k arbitrary: x rule: OrderDAG.induct)
  case (1 G k x)
    consider (inval)  $\neg$  blockDAG G | (one) blockDAG G  $\wedge$ 
      card (verts G) = 1 | (val) blockDAG G  $\wedge$ 
      card (verts G)  $\neq$  1 by auto
    then show ?case
    proof(cases)
      case inval
      then show ?thesis using 1 by auto
    next
      case one
      then show ?thesis using OrderDAG.simps 1 genesis-nodeAlt-one-sound blockDAG.is-genesis-node.simps
        using empty-set list.simps(15) singleton-iff sndI by fastforce
    next
      case val
      then show ?thesis
    proof
      have bD: blockDAG G using val by auto
      obtain M where M-in:M = choose-max-blue-set (map ( $\lambda i.$  (OrderDAG
        (reduce-past G i) k, i))
        (sorted-list-of-set (tips G))) by auto
      obtain pp where pp-in: pp = (map ( $\lambda i.$  (OrderDAG (reduce-past G i) k, i))
        (sorted-list-of-set (tips G))) using blockDAG.tips-exist by auto
      have set (snd (OrderDAG G k)) =
        set (snd (fold (app-if-blue-else-add-end G k) (top-sort G (sorted-list-of-set
        (anticone G (snd M)))))
        (add-set-list-tuple M)) unfolding M-in val using OrderDAG.simps val
        by (metis (mono-tags, lifting))
      then have set (snd (OrderDAG G k)) =
        set (top-sort G (sorted-list-of-set (anticone G (snd M))))  $\cup$  set (snd (add-set-list-tuple
        (M))
        using fold-app-mono-ex
        by (metis eq-snd-iff)
      then consider (ac) x  $\in$  set (top-sort G (sorted-list-of-set (anticone G (snd
        (M))))
        | (co) x  $\in$  set (snd (add-set-list-tuple M))
        using 1 by auto
      then show x  $\in$  verts G proof(cases)
        case ac
        then show ?thesis using top-sort-con DAG.anticone-in-verts val
          sorted-list-of-set(1) subs
          by (metis DAG.anticon-finite subsetD)
        case co
        then consider (ma) x = snd M | (nma) x  $\in$  set (snd( fst(M)))
          using add-set-list-tuple.simps
          by (metis (no-types, lifting) Un-insert-right append-Nil2 insertE
            list.simps(15) prod.collapse set-append sndI)

```

```

then show ?thesis proof(cases)
  case ma
    then show ?thesis unfolding M-in using bD
      chosen-map-simps(2) digraph.tips-in-verts subs
    by blast
  next
    have mm: choose-max-blue-set pp ∈ set pp unfolding pp-in using bD
    chosen-map-simps(4)
    by (metis (mono-tags, lifting) Nil-is-map-conv choose-max-blue-avoid-empty)

    case nma
    then have x ∈ set (snd (OrderDAG (reduce-past G (snd M)) k))
      unfolding M-in choose-max-blue-avoid-empty blockDAG.tips-not-empty
    bD
    by (metis (no-types, lifting) ex-map-conv fst-conv mm pp-in snd-conv)
    then have x ∈ verts (reduce-past G (snd M)) using 1 val chosen-map-simps
    M-in pp-in
      sorted-list-of-set(1) digraph.tips-finite subs bD
    by blast
    then show x ∈ verts G using reduce-past.simps induce-subgraph-verts
    past-nodes.simps
    by auto
    qed
  qed
  qed
  qed
  qed
  qed

```

lemma OrderDAG-length:

```

  shows blockDAG G  $\implies$  length (snd (OrderDAG G k)) = card (verts G)
proof(induct G k rule: OrderDAG.induct)
  case (1 G k)
    then show ?case proof (cases G rule: OrderDAG-casesAlt)
      case ntB
      then show ?thesis using 1 by auto
    next
      case one
      then show ?thesis using OrderDAG.simps by auto
    next
      case more
      show ?thesis using 1
    proof –
      have bD: blockDAG G using 1 by auto
      obtain ma where pp-in: ma = (choose-max-blue-set (map (λi. (OrderDAG
(reduce-past G i) k, i))
(sorted-list-of-set (tips G))))
      by (metis)
      then have backw: OrderDAG G k = fold (app-if-blue-else-add-end G k)

```

```

      (top-sort G (sorted-list-of-set (anticone G (snd ma))))
      (add-set-list-tuple ma) using OrderDAG.simps pp-in more
    by (metis (mono-tags, lifting) less-numeral-extra(4))
  have tt: snd ma ∈ set (sorted-list-of-set (tips G)) using pp-in chosen-max-tip

    more by auto
  have ttt: snd ma ∈ tips G using chosen-max-tip(2) pp-in
    more by auto
  then have bD2: blockDAG (reduce-past G (snd ma)) using blockDAG.tips-unequal-gen
bD more
    blockDAG.reduce-past-dagbased bD tips-def
  by fastforce
  then have length (snd (OrderDAG (reduce-past G (snd ma)) k))
    = card (verts (reduce-past G (snd ma)))
    using 1 tt bD2 more by auto
  then have length (snd (fst ma))
    = card (verts (reduce-past G (snd ma)))
    using bD chosen-map-simps(6) pp-in
  by fastforce
  then have length (snd (add-set-list-tuple ma)) = 1 + card (verts (reduce-past
G (snd ma)))
    by (metis add-set-list-tuple-length plus-1-eq-Suc prod.collapse)
  then show ?thesis unfolding backw
    using subs DAG.verts-size-comp ttt
    add.assoc add commute bD fold-app-length length-sorted-list-of-set top-sort-len
    by (metis (full-types))
  qed
qed
qed

```

```

lemma OrderDAG-total:
  assumes blockDAG G
  shows set (snd (OrderDAG G k)) = verts G
  using Verts-in-OrderDAG OrderDAG-in-verts assms(1)
  by blast

```

```

lemma OrderDAG-distinct:
  assumes blockDAG G
  shows distinct (snd (OrderDAG G k))
  using OrderDAG-length OrderDAG-total
    card-distinct assms
  by metis

```

end

```

theory Extend-blockDAG
  imports blockDAG
begin

```

6 Extend blockDAGs

6.1 Definitions

```

locale Append-One = blockDAG +
  fixes  $G-A :: ('a, 'b)$  pre-digraph (structure)
    and  $app :: 'a$ 
  assumes  $bD-A$ : blockDAG  $G-A$ 
    and  $app-in$ :  $app \in \text{verts } G-A$ 
    and  $app-notin$ :  $app \notin \text{verts } G$ 
    and  $GG-A$  :  $G = \text{pre-digraph.del-vert } G-A \text{ } app$ 
    and  $new-node$ :  $\forall b \in \text{verts } G-A. \neg b \rightarrow_{G-A} app$ 

```

```

locale Honest-Append-One = Append-One +
  assumes  $ref-tips$ :  $\forall t \in \text{tips } G. app \rightarrow_{G-A} t$ 

```

```

locale Append-One-Honest-Dishonest = Honest-Append-One +
  fixes  $G-AB :: ('a, 'b)$  pre-digraph (structure)
    and  $dis :: 'a$ 
  assumes  $app-two$ : Append-One  $G-A$   $G-AB$   $dis$ 
    and  $dis-n-app$ :  $\neg dis \rightarrow_{G-AB} app$ 

```

6.2 Append-One Lemmas

```

lemma (in Append-One)  $new-node-alt$ :
   $(\forall b. \neg b \rightarrow_{G-A} app)$ 
proof(auto)
  fix  $b$ 
  assume  $a2$ :  $b \rightarrow_{G-A} app$ 
  then have  $b \in \text{verts } G-A$  using  $wf-digraph.adj-in-verts(1)$   $bD-A$  subs by metis
  then show False using  $new-node$   $a2$  by auto
qed

```

```

lemma (in Append-One)  $append-subverts-leq$ :
   $\text{verts } G \subseteq \text{verts } G-A$ 
  unfolding  $GG-A$  pre-digraph.verts-del-vert by auto

```

```

lemma (in Append-One)  $append-subverts$ :
   $\text{verts } G \subset \text{verts } G-A$ 
  unfolding  $GG-A$  pre-digraph.verts-del-vert using  $app-in$   $app-notin$  by auto

```

```

lemma (in Append-One)  $append-verts$ :
   $\text{verts } G-A = \text{verts } G \cup \{app\}$ 

```

```

unfolding GG-A pre-digraph.verts-del-vert using app-in app-notin by auto

lemma (in Append-One) append-verts-in:
  assumes  $a \in \text{verts } G$ 
  shows  $a \in \text{verts } G-A$ 
  unfolding append-verts
  by (simp add: assms)

lemma (in Append-One) append-verts-diff:
  shows  $\text{verts } G = \text{verts } G-A - \{app\}$ 
  using append-verts app-in app-notin by auto

lemma (in Append-One) append-verts-cases:
  assumes  $a \in \text{verts } G-A$ 
  obtains ( $a\text{-in-}G$ )  $a \in \text{verts } G \mid (a\text{-eq-}app) \ a = app$ 
  using append-verts assms by auto

lemma (in Append-One) append-subarcs-leq:
   $\text{arcs } G \subseteq \text{arcs } G-A$ 
  unfolding GG-A pre-digraph.arcs-del-vert using app-in app-notin
  using wf-digraph-def subs Append-One-axioms by blast

lemma (in Append-One) append-subarcs:
   $\text{arcs } G \subset \text{arcs } G-A$ 
proof
  show  $\text{arcs } G \subseteq \text{arcs } G-A$  using append-subarcs-leq by simp
  obtain gen where gen-rec:  $app \rightarrow^+_{G-A} gen$  using bD-A blockDAG.genesis
  app-in
    app-notin append-subverts
    genesis-in-verts new-node psubsetD tranclE new-node-alt
  by (metis (mono-tags, lifting))
  then obtain walk where walk-in:  $\text{pre-digraph.awalk } G-A \ app \ walk \ gen \wedge \ walk$ 
 $\neq []$ 
    using wf-digraph.reachable1-awalk bD-A subs
    by metis
  then obtain e where  $\exists es. \ walk = e \ \# \ es$ 
    by (metis list.exhaust)
  then have e-in:  $e \in \text{arcs } G-A \wedge \text{tail } G-A \ e = app$ 
    using wf-digraph.awalk-simps(2)
    bD-A subs walk-in
    by metis
  then have  $e \notin \text{arcs } G$  using wf-digraph-def app-notin
    blockDAG-axioms subs GG-A pre-digraph.tail-del-vert
    by metis
  then show  $\text{arcs } G \neq \text{arcs } G-A$  using e-in by auto
qed

lemma (in Append-One) append-head:
   $\text{head } G-A = \text{head } G$ 

```

```

using GG-A
by (simp add: pre-digraph.head-del-vert)

lemma (in Append-One) append-tail:
  tail G-A = tail G
using GG-A
by (simp add: pre-digraph.tail-del-vert)

lemma (in Append-One) append-subgraph:
  subgraph G G-A
using GG-A blockDAG-axioms subs bD-A
by (simp add: subs wf-digraph.subgraph-del-vert)

lemma (in Append-One) append-induce-subgraph:
  G-A  $\upharpoonright$  (verts G) = G
proof –
  have aaa: arcs G = {e ∈ arcs G-A. tail G-A e ∈ verts G ∧ head G-A e ∈ verts G}
  unfolding GG-A pre-digraph.arcs-del-vert pre-digraph.verts-del-vert
  using append-verts bD-A subs wf-digraph-def
  by (metis (no-types, lifting) Diff-insert-absorb Un-empty-right
    Un-insert-right app-notin insertE)
  show G-A  $\upharpoonright$  verts G = G
  unfolding induce-subgraph-def
  using aaa app-notin append-head append-tail
    arcs-del-vert del-vert-def del-vert-not-in-graph verts-del-vert
  by (metis (no-types, lifting))
qed

lemma (in Append-One) append-induced-subgraph:
  induced-subgraph G G-A
proof –
  interpret W: blockDAG G-A using bD-A by auto
  show ?thesis
  using W.induced-induce append-induce-subgraph append-subverts psubsetE
  by (metis)
qed

lemma (in Append-One) append-not-reached:
   $\forall b \in \text{verts } G-A. \neg b \rightarrow^+_{G-A} \text{app}$ 
using tranclE wf-digraph.reachable1-in-verts(2) bD-A subs new-node
by metis

lemma (in Append-One) append-not-reached-all:
   $\forall b. \neg b \rightarrow^+_{G-A} \text{app}$ 
using tranclE bD-A new-node-alt

```

by *metis*

lemma (in *Append-One*) *append-not-headed*:
 $\forall b \in \text{arcs } G-A. \neg \text{head } G-A \ b = \text{app}$
proof(*rule ccontr, safe*)
 fix *b*
 assume $b \in \text{arcs } G-A$
 and $\text{app} = \text{head } G-A \ b$
 then have $\text{tail } G-A \ b \rightarrow_{G-A} \text{app}$
 unfolding *arcs-ends-def arc-to-ends-def*
 by *auto*
 then show *False*
 by (*simp add: new-node-alt*)
qed

lemma (in *Append-One*) *dominates-preserve*:
 assumes $b \in \text{verts } G$
 shows $b \rightarrow_{G-A} c \longleftrightarrow b \rightarrow_G c$
proof
 assume $b \rightarrow_G c$
 then show $b \rightarrow_{G-A} c$
 using *append-subgraph pre-digraph.adj-mono*
 by *metis*
next
 have $b\text{-napp} : b \neq \text{app}$ using *assms(1) append-verts-diff* by *auto*
 assume $bc : b \rightarrow_{G-A} c$
 then have $c\text{-napp} : c \neq \text{app}$ using *new-node-alt* by *auto*
 show $b \rightarrow_G c$ unfolding *arcs-ends-def arc-to-ends-def*
 using *GG-A pre-digraph.arcs-del-vert bc b-napp c-napp*
 using *append-head append-tail* by *fastforce*
qed

lemma (in *Append-One*) *reachable1-preserve*:
 assumes $b \in \text{verts } G$
 shows $(b \rightarrow^+_{G-A} c) \longleftrightarrow b \rightarrow^+ c$
proof(*standard*)
 assume $b \rightarrow^+ c$
 then show $b \rightarrow^+_{G-A} c$
 using *tranc1-mono append-subgraph arcs-ends-mono*
 by (*metis*)
next
 interpret *B2: blockDAG G-A* using *bD-A* by *simp*
 assume $c\text{-re} : b \rightarrow^+_{G-A} c$
 show $b \rightarrow^+ c$
 using *c-re*
proof(*cases rule: tranc1-induct*)
 case (*base y*)

```

    then have  $b \rightarrow_G y$  using dominates-preserve assms(1) by auto
    then show  $b \rightarrow^+ y$  by auto
next
fix  $y z$ 
assume  $b-y: b \rightarrow^+ y$ 
then have  $y-in: y \in \text{verts } G$  using reachable1-in-verts(2)
    by (metis)
assume  $y \rightarrow_{G-A} z$ 
then have  $y \rightarrow_G z$  using dominates-preserve y-in by auto
then show  $b \rightarrow^+ z$  using  $b-y$  by auto
qed
qed

lemma (in Append-One) append-past-nodes:
  assumes  $a \in \text{verts } G$ 
  shows  $\text{past-nodes } G \ a = \text{past-nodes } G-A \ a$ 
  unfolding past-nodes.simps append-verts using
    assms reachable1-preserve append-not-reached-all by auto

lemma (in Append-One) append-is-tip:
  is-tip G-A app
  unfolding is-tip.simps
  using app-in new-node append-not-reached
  by metis

lemma (in Append-One) append-in-tips:
   $app \in \text{tips } G-A$ 
  unfolding tips-def
  using app-in new-node append-is-tip CollectI
  by metis

lemma (in Append-One) append-future:
  assumes  $a \in \text{verts } G$ 
  shows  $\text{future-nodes } G \ a = \text{future-nodes } G-A \ a - \{app\}$ 
  unfolding future-nodes.simps append-verts
proof(auto simp: assms reachable1-preserve app-notin) qed

lemma (in Append-One) append-greater-1:
   $\text{card } (\text{verts } G-A) > 1$ 
  unfolding append-verts
  using app-notin no-empty-blockDAG by auto

lemma (in Append-One) append-reduce-some:
  assumes  $a \in \text{verts } G$ 
  shows  $\text{reduce-past } G-A \ a = \text{reduce-past } G \ a$ 
  unfolding reduce-past.simps past-nodes.simps append-head append-tail
    induce-subgraph-def append-verts
proof(standard,simp,standard)
  have  $a \neq app$  using assms(1) append-verts-diff by auto

```



```

then show  $vv: \{b. (b = \text{app} \vee b \in \text{verts } G) \wedge a \rightarrow^+_{G-A} b\} = \{b \in \text{verts } G. a \rightarrow^+ b\}$ 
using reachable1-preserve assms reachable1-in-verts(2) by blast
show  $\{e \in \text{arcs } G-A. (tail\ G\ e = \text{app} \vee tail\ G\ e \in \text{verts } G) \wedge a \rightarrow^+_{G-A} tail\ G\ e \wedge (head\ G\ e = \text{app} \vee head\ G\ e \in \text{verts } G) \wedge a \rightarrow^+_{G-A} head\ G\ e\} = \{e \in \text{arcs } G. tail\ G\ e \in \text{verts } G \wedge a \rightarrow^+ tail\ G\ e \wedge head\ G\ e \in \text{verts } G \wedge a \rightarrow^+ head\ G\ e\}$ 
unfolding vv GG-A pre-digraph.arcs-del-vert using append-not-reached-all
using GG-A append-head append-tail assms reachable1-preserve by fastforce
qed

```

```

lemma blockDAG-induct-append[consumes 1, case-names base step]:
  assumes fund: blockDAG G
  assumes cases:  $\bigwedge V::('a,'b)$  pre-digraph. blockDAG V  $\implies$  P (blockDAG.gen-graph V)
   $\bigwedge G\ G-A::('a,'b)$  pre-digraph.  $\bigwedge \text{app}::'a.$ 
    Append-One G G-A app
     $\implies (P\ G)$ 
     $\implies P\ G-A$ 
  shows P G
  using assms
proof(induct G rule: blockDAG-induct)
  case (base V)
  then show ?case by auto
next
  case (step H)
  show ?case proof(cases H rule: blockDAG.blockDAG-cases2, simp add: step)
  case 2
  then have blockDAG (blockDAG.gen-graph H) using step(2) blockDAG.gen-graph-sound
by auto
  then show ?thesis using step(3) 2 by metis
next
  case 3
  then obtain Ha and b where bD-Ha: blockDAG Ha and b-in: b  $\in$  verts H
  and del-v: Ha = pre-digraph.del-vert H b and nre:  $(\forall c \in \text{verts } H. (c, b) \notin (\text{arcs-ends } H)^+)$ 
  by auto
  then have b  $\notin$  verts Ha unfolding del-v pre-digraph.verts-del-vert by auto
  then have Append-One Ha H b unfolding Append-One-def Append-One-axioms-def

  using bD-Ha b-in del-v nre step.hyps(2) by auto
  then show ?thesis using step
  using bD-Ha b-in del-v by auto
qed
qed

```

6.3 Honest-Append-One Lemmas

lemma (in *Honest-Append-One*) *reaches-all*:

$\forall v \in \text{verts } G. \text{app} \rightarrow^+_{G-A} v$

proof

fix v

assume $v\text{-in}: v \in \text{verts } G$

consider $\text{is-tip } G \ v \mid \neg \text{is-tip } G \ v$ **by** *auto*

then show $\text{app} \rightarrow^+_{G-A} v$

proof(*cases*)

case 1

then show *?thesis* **using** *ref-tips v-in is-tip.simps r-into-trancl'* *reached-by*
by (*metis*)

next

case 2

then have $v \notin \text{tips } G$ **unfolding** *tips-def* **by** *simp*

then obtain t **where** $t\text{-in}: t \in \text{tips } G \wedge t \rightarrow^+_{G-A} v$

using *reached-by-tip v-in* **by** *auto*

then have $t \rightarrow^+_{G-A} v$ **using** *v-in append-subgraph arcs-ends-mono trancl-mono*
by (*metis*)

moreover have $\text{app} \rightarrow^+_{G-A} t$

using *ref-tips v-in r-into-trancl' t-in*

by (*metis*)

ultimately show *?thesis* **using** *trancl-trans* **by** *auto*

qed

qed

lemma (in *Honest-Append-One*) *append-past-all*:

past-nodes $G-A \text{app} = \text{verts } G$

unfolding *past-nodes.simps append-verts*

using *reaches-all DAG.cycle-free bD-A subs*

by *fastforce*

lemma (in *Honest-Append-One*) *append-in-future*:

assumes $a \in \text{verts } G$

shows $\text{app} \in \text{future-nodes } G-A \ a$

unfolding *future-nodes.simps append-verts*

using *assms*

proof(*auto simp: reaches-all reachable1-preserve*) **qed**

lemma (in *Honest-Append-One*) *append-is-only-tip*:

$\text{tips } G-A = \{\text{app}\}$

proof *safe*

show $\text{app} \in \text{tips } G-A$ **using** *append-in-tips* **by** *simp*

fix x

assume *as1*: $x \in \text{tips } G-A$

then have $x\text{-in}: x \in \text{verts } G-A$ **using** *digraph.tips-in-verts bD-A subs* **by** *auto*

show $x = \text{app}$

```

proof(rule ccontr)
  assume  $x \neq \text{app}$ 
  then have  $x \in \text{verts } G$  using append-verts x-in by auto
  then have  $\text{app} \rightarrow^+_{G-A} x$  using reaches-all by auto
  then have  $\neg \text{is-tip } G-A \ x$  unfolding is-tip.simps using app-in by auto
  then show False using as1 CollectD unfolding tips-def by auto
qed
qed

lemma (in Honest-Append-One) reduce-append:
  reduce-past G-A app = G
  unfolding reduce-past.simps past-nodes.simps
proof –
  have  $\{b \in \text{verts } G. \text{app} \rightarrow^+_{G-A} b\} = \text{verts } G$ 
    using reaches-all by auto
  moreover have  $\{b \in \text{verts } G. \text{app} \rightarrow^+_{G-A} b\} = \{b \in \text{verts } G-A. \text{app} \rightarrow^+_{G-A} b\}$ 
qed
  unfolding append-verts using append-is-tip by fastforce
  ultimately have  $\{b \in \text{verts } G-A. \text{app} \rightarrow^+_{G-A} b\} = \text{verts } G$  by simp
  then show  $G-A \upharpoonright \{b \in \text{verts } G-A. \text{app} \rightarrow^+_{G-A} b\} = G$ 
    unfolding induce-subgraph-def
    using append-induced-subgraph induced-subgraph-def
    append-head append-tail
    by (metis (no-types, lifting) Collect-cong app-notin arcs-del-vert
      del-vert-def del-vert-not-in-graph verts-del-vert)
qed

lemma (in Honest-Append-One) append-no-anticone:
  anticone G-A app = {}
  unfolding anticone.simps
proof safe
  fix  $x$ 
  assume  $x \in \text{verts } G-A$ 
  and  $\text{app} \neq x$ 
  and as:  $(\text{app}, x) \notin (\text{arcs-ends } G-A)^+$ 
  then have  $x \in \text{verts } G$ 
    using append-verts by auto
  then have  $\text{app} \rightarrow^+_{G-A} x$ 
    using reaches-all by auto
  then show  $x \rightarrow^+_{G-A} \text{app}$ 
    using as by auto
qed

```

6.4 Honest-Dishonest-Append-One Lemmas

```

lemma (in Append-One-Honest-Dishonest) app-dis-not-reached:
  shows  $\neg \text{dis} \rightarrow^+_{G-AB} \text{app}$ 
proof(rule ccontr, cases dis app (arcs-ends G-AB) rule: converse-trancl-induct,
  auto)

```

```

fix y
assume y-d:  $y \rightarrow_{G-AB} app$ 
interpret D1: Append-One G-A G-AB dis using app-two by simp
interpret B3: blockDAG G-AB using D1.bD-A by auto
have  $y \in \text{verts } G-AB$  using y-d B3.wellformed by auto
then consider  $y = dis \mid y \in \text{verts } G-A$  using D1.append-verts by auto
then show False
proof(cases)
  case 1
  then have  $dis \rightarrow_{G-AB} app$  using y-d by auto
  then show ?thesis using dis-n-app by auto
next
  case 2
  then have  $y \rightarrow^+_{G-A} app$  using D1.reachable1-preserve y-d by blast
  then show ?thesis using append-not-reached-all by auto
qed
qed

```

lemma (in Append-One-Honest-Dishonest) app-dis-only-tips:

$\text{tips } G-AB = \{app, dis\}$

proof safe

interpret A2: Append-One G-A G-AB dis using app-two by auto

show $dis \in \text{tips } G-AB$ using A2.append-in-tips by simp

show $app \in \text{tips } G-AB$ unfolding tips-def is-tip.simps A2.append-verts
using app-dis-not-reached reachable1-preserve append-not-reached
A2.reachable1-preserve app-in

by simp

fix x

assume as1: $x \in \text{tips } G-AB$

and app-x: $x \neq app$

have $x \notin \text{verts } G$

proof

assume x-vG: $x \in \text{verts } G$

then have $x \in \text{verts } G-A$ using append-verts by auto

then have $app \rightarrow^+_{G-AB} x$ using A2.reachable1-preserve reaches-all x-vG

app-in

by simp

then have $x \notin \text{tips } G-AB$

by (metis A2.append-verts-in app-in is-tip.elims(2) tips-tips)

then show False using as1 by simp

qed

then show $x = dis$ unfolding

append-verts-diff A2.append-verts-diff using app-x as1 tip-in-verts by force

qed

lemma (in Append-One-Honest-Dishonest) app-not-dis:

$app \neq dis$

using app-in Append-One.app-notin app-two by metis

```

lemma (in Append-One-Honest-Dishonest) app-in2:
  app ∈ verts G-AB
  using app-in Append-One.append-verts-in app-two by metis

context Append-One-Honest-Dishonest

begin
interpretation A2: Append-One G-A G-AB dis using local.app-two by auto

lemma app2-head:
  head G-AB = head G using append-head A2.append-head by simp

lemma app2-tail:
  tail G-AB = tail G using append-tail A2.append-tail by simp

lemma app-in-future2:
  assumes a ∈ verts G
  shows app ∈ future-nodes G-AB a
  unfolding future-nodes.simps A2.append-verts
  using app-in A2.reachable1-preserve app-in2 assms reaches-all
  by simp

lemma append-past-nodes2:
  past-nodes G-AB app = verts G
  using app-in A2.append-past-nodes append-past-all
  by auto

lemma reachable1-preserve2:
  assumes b ∈ verts G
  shows (b →+G-AB c) ↔ b →+ c
  using A2.reachable1-preserve append-verts-in reachable1-preserve assms by auto

lemma reaches-all-in-G:
  assumes b ∈ verts G
  shows app →+G-AB b
  using assms(1) reaches-all A2.reachable1-preserve app-in
  by simp

lemma append-induce-subgraph2:
  G-AB ↑ (verts G) = G
  using append-induce-subgraph A2.append-induce-subgraph
  unfolding append-verts
  by (metis A2.append-past-nodes Append-One.append-reduce-some
    app-in app-two append-past-all reduce-past.elims)

```

lemma *append-induced-subgraph2*:
induced-subgraph G $G-AB$
using *wf-digraph.induced-induce* $A2.bD-A$ *subs* *append-induce-subgraph2* *append-subverts-leq*
 $A2.append-subverts-leq$ *subset-trans*
by *metis*

lemma *reduce-append2*:
reduce-past $G-AB$ *app* = G
unfolding *reduce-past.simps* *past-nodes.simps*
proof –
have $\{b \in \text{verts } G. \text{ app } \rightarrow^+_{G-AB} b\} = \text{verts } G$
using *reaches-all-in-G* **by** *auto*
moreover have $\{b \in \text{verts } G. \text{ app } \rightarrow^+_{G-AB} b\} = \{b \in \text{verts } G-AB. \text{ app } \rightarrow^+_{G-AB} b\}$
unfolding $A2.append-verts$ *append-verts* **using** *append-is-tip* *app-dis-not-reached*
app-dis-only-tips
 $A2.append-not-reached-all$ $A2.reachable1-preserve$ **by** *fastforce*
ultimately have $\{b \in \text{verts } G-AB. \text{ app } \rightarrow^+_{G-AB} b\} = \text{verts } G$ **by** *simp*
then show $G-AB \upharpoonright \{b \in \text{verts } G-AB. \text{ app } \rightarrow^+_{G-AB} b\} = G$
using *append-induced-subgraph2*
unfolding *induce-subgraph-def* *induced-subgraph-def* *app2-head* *app2-tail*
by (*metis* (*no-types*, *lifting*) *Collect-cong* *app-notin* *del-vert-def* *del-vert-not-in-graph*
 $pre-digraph.select-convs(2)$ *verts-del-vert*)

qed

end

end

theory *Properties*

imports *blockDAG* *Extend-blockDAG* *Spectre* *Ghostdag*

begin

definition *Linear-Order*:: $((a, b) \text{ pre-digraph} \Rightarrow a \text{ rel}) \Rightarrow \text{bool}$
where *Linear-Order* $A \equiv (\forall G. \text{ blockDAG } G \longrightarrow \text{linear-order-on } (\text{verts } G) (A \text{ } G))$

definition *Order-Preserving*:: $((a, b) \text{ pre-digraph} \Rightarrow a \text{ rel}) \Rightarrow \text{bool}$
where *Order-Preserving* $A \equiv (\forall G \ a \ b. \text{ blockDAG } G \longrightarrow a \rightarrow^+_G b \longrightarrow (b, a) \in (A \text{ } G))$

definition *One-Appending-Monotone*:: $((a, b) \text{ pre-digraph} \Rightarrow a \text{ rel}) \Rightarrow \text{bool}$
where *One-Appending-Monotone* $A \equiv$

$(\forall G \ G\text{-}A \ a \ b \ c. \text{Honest-Append-One } G \ G\text{-}A \ a \longrightarrow ((b,c) \in (A \ G) \longrightarrow (b,c) \in (A \ G\text{-}A)))$

definition *One-Appending-Robust*:: $((a,b) \text{ pre-digraph} \Rightarrow a \text{ rel}) \Rightarrow \text{bool}$
where *One-Appending-Robust* $A \equiv$
 $(\forall G \ G\text{-}A \ G\text{-}AB \ a \ b \ c \ d. \text{Append-One-Honest-Dishonest } G \ G\text{-}A \ a \ G\text{-}AB \ b$
 $\longrightarrow ((c,d) \in (A \ G) \longrightarrow (c,d) \in (A \ G\text{-}AB)))$

end
theory *Spectre-Properties*
imports *Spectre Extend-blockDAG Properties*
begin

7 SPECTRE properties

7.1 SPECTRE Order Preserving

lemma *vote-Spectre-Preserving*:
assumes $c \rightarrow^+_G b$
shows $\text{vote-Spectre } G \ a \ b \ c \in \{0,1\}$
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.premis* *trancl-trans*)
next
case *three*
then have *a-not-gen*: $\neg \text{blockDAG.is-genesis-node } V \ a$
using *blockDAG.genesis-reaches-nothing*
by *metis*
then have *bD*: $\text{blockDAG (reduce-past } V \ a)$ **using** *blockDAG.reduce-past-dagbased*

three **by** *auto*

have $b\text{-in}2$: $b \in \text{past-nodes } V a$ **using** *three* **by** *auto*
also have $c\text{-in}2$: $c \in \text{past-nodes } V a$ **using** *three* **by** *auto*
ultimately have $c \rightarrow^+ \text{reduce-past } V a b$ **using** *DAG.reduce-past-path2 three 1*
by (*metis blockDAG.axioms(1)*)
then have *allsorted01*: $\forall x. x \in \text{set } (\text{sorted-list-of-set } (\text{past-nodes } V a)) \longrightarrow$
 $\text{vote-Spectre } (\text{reduce-past } V a) x b c \in \{0, 1\}$ **using** *1 three* **by** *auto*
then have *all01*: $\forall x. x \in (\text{past-nodes } V a) \longrightarrow$
 $\text{vote-Spectre } (\text{reduce-past } V a) x b c \in \{0, 1\}$
using *subs three sorted-list-of-set(1) DAG.finite-past*
by *metis*
obtain *wit* **where** *wit-in*: $\text{wit} \in \text{past-nodes } V a$
and *wit-vote*: $\text{vote-Spectre } (\text{reduce-past } V a) \text{wit} b c \neq 0$
using *vote-Spectre-one-exists b-in2 c-in2 bD induce-subgraph-verts reduce-past.simps*
by *metis*
then have *wit-vote1*: $\text{vote-Spectre } (\text{reduce-past } V a) \text{wit} b c = 1$ **using** *all01*
by *blast*
obtain *the-map* **where** *the-map-in*:
 $\text{the-map} = (\text{map } (\lambda i. \text{vote-Spectre } (\text{reduce-past } V a) i b c)$
 $(\text{sorted-list-of-set } (\text{past-nodes } V a)))$
by *auto*
have *all01-1*: $\forall x \in \text{set } \text{the-map}. x \in \{0, 1\}$
unfolding *the-map-in set-map*
using *allsorted01* **by** *blast*
have $\exists x \in \text{set } \text{the-map}. x = 1$
unfolding *the-map-in set-map*
using *wit-in wit-vote1*
 $\text{sorted-list-of-set}(1) \text{ DAG.finite-past } bD \text{ subs}$
by (*metis (no-types, lifting) image-iff three*)
then have $\exists x \in \text{set } \text{the-map}. x > 0$
using *zero-less-one* **by** *blast*
moreover have $\forall x \in \text{set } \text{the-map}. x \geq 0$ **using** *all01-1*
by (*metis empty-iff insert-iff less-int-code(1) not-le-imp-less zero-le-one*)
ultimately have $\text{signum } (\text{sum-list } \text{the-map}) = 1$ **using** *sumlist-one-mono* **by**
simp
then have $\text{tie-break-int } b c (\text{signum } (\text{sum-list } \text{the-map})) = 1$ **using** *tie-break-int.simps*
by *simp*
then have $\text{vote-Spectre } V a b c = 1$ **unfolding** *the-map-in* **using** *three*
vote-Spectre.simps
by *simp*
then show *?thesis* **by** *simp*
next
case *four*
then have *all01*: $\forall a2. a2 \in \text{set } (\text{sorted-list-of-set } (\text{future-nodes } V a)) \longrightarrow$
 $\text{vote-Spectre } V a2 b c \in \{0, 1\}$
using *1*
by *metis*
have $\forall a2. a2 \in \text{set } (\text{sorted-list-of-set } (\text{future-nodes } V a))$
 $\longrightarrow \text{vote-Spectre } V a2 b c \geq 0$
proof *safe*


```

    fix a2
    assume a2 ∈ set (sorted-list-of-set (future-nodes V a))
    then have vote-Spectre V a2 b c ∈ {0, 1} using all01 by auto
    then show vote-Spectre V a2 b c ≥ 0
      by fastforce
    qed
    then have (sum-list (map (λi. vote-Spectre V i b c)
      (sorted-list-of-set (future-nodes V a)))) ≥ 0 using sum-list-mono sum-list-0
      by metis
    then have signum (sum-list (map (λi. vote-Spectre V i b c)
      (sorted-list-of-set (future-nodes V a)))) ∈ {0, 1} unfolding signum.simps
      by simp
    then show ?thesis using four by simp
  qed
qed

```

lemma *Spectre-Order-Preserving*:

```

  assumes blockDAG G
  and b →+G a
  shows Spectre-Order G a b
  proof –
    interpret B: blockDAG G using assms(1) by auto
    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 ∈ verts G using B.reachable1-in-verts(2) assms(2)
      by metis
    have b-in: b ∈ verts G using B.reachable1-in-verts(1) assms(2)
      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 ∈ verts G and wit-vote: vote-Spectre G wit a b ≠
    0
      using vote-Spectre-one-exists a-in b-in assms(1)
      by blast
    have (vote-Spectre G wit a b) ∈ set the-map
      unfolding the-map-in set-map
      using B.blockDAG-axioms fin-digraph.finite-verts
      subs sorted-list-of-set(1) wit-in image-iff
      by metis
    then have exune: ∃ x ∈ set the-map. x ≠ 0
      using wit-vote by blast
    have all01: ∀ x ∈ set the-map. x ∈ {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 ∃ x ∈ set the-map. x = 1 using exune

```

```

    by blast
  then have  $\exists x \in \text{set the-map. } x > 0$ 
    using zero-less-one by blast
  moreover have  $\forall x \in \text{set the-map. } x \geq 0$  using all01
    by (metis empty-iff insert-iff less-int-code(1) not-le-imp-less zero-le-one)
  ultimately have  $\text{signum (sum-list the-map)} = 1$  using sumlist-one-mono by
simp
  then have tie-break-int a b ( $\text{signum (sum-list the-map)}$ ) = 1 using tie-break-int.simps
    by simp
  then show ?thesis unfolding the-map-in Spectre-Order-def by simp
qed

```

```

lemma SPECTRE-Preserving:
  assumes blockDAG G
    and  $b \rightarrow^+_G a$ 
  shows  $(a,b) \in (\text{SPECTRE } G)$ 
  unfolding SPECTRE-def
  using assms wf-digraph.reachable1-in-verts subs
    Spectre-Order-Preserving
    SigmaI case-prodI mem-Collect-eq by fastforce

```

```

lemma Order-Preserving SPECTRE
  unfolding Order-Preserving-def
  using SPECTRE-Preserving by auto

```

```

lemma vote-Spectre-antisymmetric:
  shows  $b \neq c \implies \text{vote-Spectre } V a b c = - (\text{vote-Spectre } V a c b)$ 
proof(induction V a b c rule: vote-Spectre.induct)
  case (1 V a b c)
  show  $\text{vote-Spectre } V a b c = - \text{vote-Spectre } V a c b$ 
  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:  $\text{vote-Spectre } V a b c = \text{tie-break-int } b c (\text{signum (sum-list (map$ 
( $\lambda i.$ 
 $(\text{vote-Spectre (reduce-past } V a) i b c)) (\text{sorted-list-of-set (past-nodes } V a))))))$ 
      using vote-Spectre.simps

```

```

    by (metis (mono-tags, lifting))
  have ff1: vote-Spectre V a c b = tie-break-int c b (signum (sum-list (map (λi.
    (vote-Spectre (reduce-past V a) i c b)) (sorted-list-of-set (past-nodes V a)))))
    using vote-Spectre.simps three by fastforce
  then have ff2: vote-Spectre V a c b = tie-break-int c b (signum (sum-list (map
    (λi.
      (− vote-Spectre (reduce-past V a) i b c)) (sorted-list-of-set (past-nodes V a)))))
    using three 1 map-eq-conv ff
    by (smt (verit, best))
  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 = − (tie-break-int b c (signum (sum-list (map
    (λi.
      (vote-Spectre (reduce-past V a) i b c)) (sorted-list-of-set (past-nodes V a)))))
    using antisymmetric-sumlist 1 ff2 antisymmetric-signum antisymmetric-tie-break
    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 = signum (sum-list (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))
  then have ff2: vote-Spectre V a c b = signum (sum-list (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 = − ( signum (sum-list (map (λi.
    (vote-Spectre V i b c)) (sorted-list-of-set (future-nodes V a)))))
    using antisymmetric-sumlist 1 ff2 antisymmetric-signum
    by (metis verit-minus-simplify(4))
  then show ?thesis using ff
    by linarith
qed
qed

```

lemma *vote-Spectre-reflexive:*

assumes *blockDAG V*

and $a \in \text{verts } V$

shows $\forall b \in \text{verts } V. \text{ vote-Spectre } V b a a = 1$ **using** *vote-Spectre.simps assms*

by *auto*

lemma *Spectre-Order-reflexive*:

assumes *blockDAG V*

and $a \in \text{verts } V$

shows *Spectre-Order V a a*

unfolding *Spectre-Order-def*

proof –

obtain l **where** $l\text{-def}: l = (\text{map } (\lambda i. \text{vote-Spectre } V i a a) (\text{sorted-list-of-set } (\text{verts } V)))$

by *auto*

have *only-one*: $l = (\text{map } (\lambda i. 1) (\text{sorted-list-of-set } (\text{verts } V)))$

using $l\text{-def}$ *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)* $l\text{-def}$)

have *sum-list* $l = \text{card } (\text{verts } V)$ **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 *tie-break-int a a*

$(\text{signum } (\text{sum-list } (\text{map } (\lambda i. \text{vote-Spectre } V i a a) (\text{sorted-list-of-set } (\text{verts } V)))) = 1$

using $l\text{-def}$ *ne* *tie-break-int.simps*

list.exhaust *verit-comp-simplify1(1)* **by** *auto*

qed

lemma *Spectre-Order-antisym*:

assumes *blockDAG V*

and $a \in \text{verts } V$

and $b \in \text{verts } V$

and $a \neq b$

shows *Spectre-Order V a b* $= (\neg (\text{Spectre-Order } V b a))$

proof –

obtain *wit* **where** *wit-in*: *vote-Spectre V wit a b* $\neq 0 \wedge \text{wit} \in \text{verts } V$

using *vote-Spectre-one-exists* *assms*

by *blast*

obtain l **where** $l\text{-def}: l = (\text{map } (\lambda i. \text{vote-Spectre } V i a b) (\text{sorted-list-of-set } (\text{verts } V)))$

by *auto*

have $\text{wit} \in \text{set } (\text{sorted-list-of-set } (\text{verts } V))$

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* $\in \text{set } l$ **unfolding** $l\text{-def}$

by (*metis* (*mono-tags*, *lifting*) *image-eqI* *list.set-map*)

then have *dm*: *tie-break-int a b* $(\text{signum } (\text{sum-list } l)) \in \{-1, 1\}$

```

    by auto
  obtain l2 where l2-def: l2 = (map (λi. 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)
  have anti: tie-break-int a b (signum (sum-list l)) =
    - tie-break-int b a (signum (sum-list l2)) unfolding minus
    using antisymmetric-sumlist antisymmetric-tie-break antisymmetric-signum
    assms(4) by metis
  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 ∈ verts V ∧ b ∈ verts V
  shows Spectre-Order V a b ∨ Spectre-Order V b a
proof safe
  assume notB: ¬ Spectre-Order V b a
  consider (eq) a = b | (neq) a ≠ 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-total:
  assumes blockDAG G
  shows total-on (verts G) (SPECTRE G)
  unfolding total-on-def SPECTRE-def
  using Spectre-Order-total assms
  by fastforce

```

```

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

```

```

lemma SPECTRE-antisym:

```

```

assumes blockDAG G
shows antisym (SPECTRE G)
unfolding antisym-def SPECTRE-def
using Spectre-Order-antisym assms by fastforce

lemma Spectre-equals-vote-Spectre-honest:
  assumes Honest-Append-One G G-A a
    and b ∈ verts G
    and c ∈ verts G
  shows Spectre-Order G b c  $\longleftrightarrow$  vote-Spectre G-A a b c = 1
proof –
  interpret H: Append-One G G-A a using assms(1) Honest-Append-One-def by
metis
  have b-in: b ∈ verts G-A using H.append-verts-in assms(2) by metis
  have c-in: c ∈ verts G-A using H.append-verts-in assms(3) by metis
  have re-all:  $\forall v \in \text{verts } G. a \rightarrow^+_{G-A} v$  using Honest-Append-One.reaches-all
assms(1) by metis
  then have r-ab:  $a \rightarrow^+_{G-A} b$  using assms(2) by simp
  have r-ac:  $a \rightarrow^+_{G-A} c$  using re-all assms(3) by simp
  consider (b-c-eq) b = c | (not-b-c-eq)  $\neg b = c$  by auto
  then show ?thesis
  proof(cases)
    case b-c-eq
    then have Spectre-Order G b c using Spectre-Order-reflexive Append-One-def
      H.Append-One-axioms assms
    by metis
    moreover have vote-Spectre G-A a b c = 1 using vote-Spectre-reflexive
      using H.bD-A H.app-in b-in b-c-eq by metis
    ultimately show ?thesis by simp
  next
    case not-b-c-eq
    then have vote-Spectre G-A a b c = (tie-break-int b c (signum (sum-list (map
      (λi.
        (vote-Spectre (reduce-past G-A a) i b c)) (sorted-list-of-set (past-nodes G-A a)))))))
      using vote-Spectre.simps Append-One.bD-A Honest-Append-One-def assms
r-ab r-ac
      Append-One.app-in b-in c-in
      map-eq-conv by fastforce
    then have the-eq: vote-Spectre G-A a b c = (tie-break-int b c (signum (sum-list
      (map (λi.
        (vote-Spectre G i b c)) (sorted-list-of-set (verts G)))))))
      using Honest-Append-One.reduce-append Honest-Append-One.append-past-all
assms(1) by metis
    show ?thesis
    unfolding the-eq Spectre-Order-def by simp
  qed
qed

```

```

lemma Spectre-Order-Appending-Mono:
  assumes Honest-Append-One  $G$   $G-A$  app
    and  $a \in \text{verts } G$ 
    and  $b \in \text{verts } G$ 
    and  $c \in \text{verts } G$ 
    and Spectre-Order  $G$   $b$   $c$ 
  shows  $\text{vote-Spectre } G$   $a$   $b$   $c \leq \text{vote-Spectre } G-A$   $a$   $b$   $c$ 
  using assms
proof(induction  $G$   $a$   $b$   $c$  rule: vote-Spectre.induct)
  case ( $1$   $G$   $a$   $b$   $c$ )
  interpret HB1: Honest-Append-One  $G$  using  $1(3)$ 
    by metis
  interpret B2: blockDAG  $G-A$  using HB1.bD-A
    by metis
  have a-in-G-A:  $a \in \text{verts } G-A$  using  $1(4)$  HB1.append-verts-in by simp
  have b-in-G-A:  $b \in \text{verts } G-A$  using  $1(5)$  HB1.append-verts-in by simp
  have c-in-G-A:  $c \in \text{verts } G-A$  using  $1(6)$  HB1.append-verts-in by simp
  show  $\text{vote-Spectre } G$   $a$   $b$   $c \leq \text{vote-Spectre } G-A$   $a$   $b$   $c$ 
  proof(cases  $a$   $b$   $c$   $G$  rule:Spectre-casesAlt)
    case no-bD
      then show ?thesis using HB1.bD  $1(4,5,6)$  by auto
    next
      case equal
        then show  $\text{vote-Spectre } G$   $a$   $b$   $c \leq \text{vote-Spectre } G-A$   $a$   $b$   $c$ 
          using B2.blockDAG-axioms a-in-G-A b-in-G-A c-in-G-A by auto
        next
          case one
            then have  $(a \rightarrow^+_{G-A} b \vee a = b) \wedge (\neg a \rightarrow^+_{G-A} c)$  using HB1.reachable1-preserve
            by auto
            then show ?thesis using one B2.blockDAG-axioms a-in-G-A b-in-G-A c-in-G-A
            by auto
          next
            case two
              then have  $\neg((a \rightarrow^+_{G-A} b \vee a = b) \wedge (\neg a \rightarrow^+_{G-A} c))$  using HB1.reachable1-preserve
              by auto
              moreover have  $(a \rightarrow^+_{G-A} c \vee a = c) \wedge (\neg a \rightarrow^+_{G-A} b)$ 
                using two HB1.reachable1-preserve by auto
              ultimately show ?thesis using two by auto
            next
              have ppp:  $\text{past-nodes } G-A$   $a = \text{past-nodes } G$   $a$  using  $1(4)$  HB1.append-past-nodes
              by auto
              have rrp:  $\text{reduce-past } G-A$   $a = \text{reduce-past } G$   $a$  using  $1(4)$  HB1.append-reduce-some
              by auto
              case three
                then have ins:  $\text{vote-Spectre } G$   $a$   $b$   $c = (\text{tie-break-int } b$   $c$   $(\text{signum } (\text{sum-list}$ 
 $(\text{map } (\lambda i.$ 
 $(\text{vote-Spectre } (\text{reduce-past } G$   $a) i$   $b$   $c))$   $(\text{sorted-list-of-set } (\text{past-nodes } G$   $a))))))$ 
                by auto

```

```

have  $\neg((a \rightarrow^+_{G-A} b \vee a = b) \wedge (\neg a \rightarrow^+_{G-A} c))$  using HB1.reachable1-preserve
three by auto
  moreover have  $\neg((a \rightarrow^+_{G-A} c \vee a = c) \wedge (\neg a \rightarrow^+_{G-A} b))$ 
    using three HB1.reachable1-preserve by auto
  moreover have  $a \rightarrow^+_{G-A} b \wedge a \rightarrow^+_{G-A} c$  using three HB1.reachable1-preserve
by auto
  ultimately have  $\text{vote-Spectre } G-A \ a \ b \ c = (\text{tie-break-int } b \ c \ (\text{signum} \ (\text{sum-list}$ 
     $(\text{map } (\lambda i.$ 
       $(\text{vote-Spectre } (\text{reduce-past } G-A \ a) \ i \ b \ c)) \ (\text{sorted-list-of-set } (\text{past-nodes } G-A \ a))))))$ 
    using three a-in-G-A b-in-G-A c-in-G-A B2.blockDAG-axioms by auto
  then have ins-2:  $\text{vote-Spectre } G-A \ a \ b \ c = (\text{tie-break-int } b \ c \ (\text{signum} \ (\text{sum-list}$ 
     $(\text{map } (\lambda i.$ 
       $(\text{vote-Spectre } (\text{reduce-past } G \ a) \ i \ b \ c)) \ (\text{sorted-list-of-set } (\text{past-nodes } G \ a))))))$ 
    unfolding ppp rrp by auto
  show ?thesis unfolding ins ins-2 by simp
next
  case four
  then have ins:  $\text{vote-Spectre } G \ a \ b \ c = \text{signum} \ (\text{sum-list} \ (\text{map } (\lambda i.$ 
     $(\text{vote-Spectre } G \ i \ b \ c)) \ (\text{sorted-list-of-set } (\text{future-nodes } G \ a))))$ 
    by auto
  have  $\neg((a \rightarrow^+_{G-A} b \vee a = b) \wedge (\neg a \rightarrow^+_{G-A} c))$  using HB1.reachable1-preserve
four by auto
  moreover have  $\neg((a \rightarrow^+_{G-A} c \vee a = c) \wedge (\neg a \rightarrow^+_{G-A} b))$ 
    using four HB1.reachable1-preserve by auto
  moreover have  $\neg(a \rightarrow^+_{G-A} b \wedge a \rightarrow^+_{G-A} c)$  using four HB1.reachable1-preserve
by auto
  ultimately have ins2:
     $\text{vote-Spectre } G-A \ a \ b \ c = \text{signum} \ (\text{sum-list} \ (\text{map } (\lambda i.$ 
       $(\text{vote-Spectre } G-A \ i \ b \ c)) \ (\text{sorted-list-of-set } (\text{future-nodes } G-A \ a))))$ 
    using B2.blockDAG-axioms a-in-G-A b-in-G-A c-in-G-A vote-Spectre.simps
four by auto
  have  $\bigwedge x . x \in \text{set} \ (\text{sorted-list-of-set } (\text{future-nodes } G \ a)) \implies x \in \text{verts } G$ 
     $\implies \text{vote-Spectre } G \ x \ b \ c \leq \text{vote-Spectre } G-A \ x \ b \ c$ 
    using 1(2,3) four
    1.prem5(5) by blast

  then have fut:  $\forall x \in \text{set} \ (\text{sorted-list-of-set } (\text{future-nodes } G \ a)).$ 
     $(\text{vote-Spectre } G \ x \ b \ c) \leq (\text{vote-Spectre } G-A \ x \ b \ c)$ 
    using HB1.finite-past HB1.past-nodes-verts
    by simp
  have futN:  $\text{future-nodes } G \ a = \text{future-nodes } G-A \ a - \{app\}$  using HB1.append-future
1(4) by auto
  have appfut:  $app \in \text{future-nodes } G-A \ a$  using HB1.append-in-future 1(4) by
auto
  have bbb:  $\text{sum-list} \ (\text{map } (\lambda i.$ 
     $(\text{vote-Spectre } G-A \ i \ b \ c)) \ (\text{sorted-list-of-set } (\text{future-nodes } G \ a))))$ 
     $= \text{sum-list} \ (\text{map } (\lambda i.$ 
     $(\text{vote-Spectre } G-A \ i \ b \ c)) \ (\text{sorted-list-of-set } (\text{future-nodes } G-A \ a))))$ 
     $- (\text{vote-Spectre } G-A \ app \ b \ c)$ 

```



```

unfolding futN
using append-diff-sorted-set B2.finite-future appfut
by blast
have vote-Spectre G-A app b c = 1
using 1.premS Spectre-equals-vote-Spectre-honest
by blast
then have sp1: sum-list (map ( $\lambda i.$ 
  (vote-Spectre G-A i b c)) (sorted-list-of-set (future-nodes G-A a))) =
  sum-list (map ( $\lambda i.$ 
  (vote-Spectre G-A i b c)) (sorted-list-of-set (future-nodes G a))) + 1
using bbb by auto
have sp2: sum-list (map ( $\lambda i.$ (vote-Spectre G i b c))
  (sorted-list-of-set (future-nodes G a)))
   $\leq$  sum-list (map ( $\lambda i.$ 
  (vote-Spectre G-A i b c)) (sorted-list-of-set (future-nodes G a)))
by (metis fut sum-list-mono)
show ?thesis unfolding ins ins2
proof (rule signum-mono)
show ( $\sum i \leftarrow \text{sorted-list-of-set (future-nodes G a). vote-Spectre G i b c}$ )
 $\leq$  ( $\sum i \leftarrow \text{sorted-list-of-set (future-nodes G-A a). vote-Spectre G-A i b c}$ )
unfolding sp1 using sp2
by linarith
qed
qed
qed

lemma One-Appending-Monotone SPECTRE
unfolding One-Appending-Monotone-def
proof safe
fix G G-A::('a::linorder,'b) pre-digraph and app b c::'a
assume Honest-Append-One G G-A app
and bcS: (b, c)  $\in$  SPECTRE G
then interpret H1: Honest-Append-One G G-A app by auto
interpret B1: blockDAG G-A using H1.bD-A by auto
have b-in: b  $\in$  verts G
and c-in: c  $\in$  verts G using bcS unfolding SPECTRE-def by auto
then have b-in2: b  $\in$  verts G-A
and c-in2: c  $\in$  verts G-A using H1.append-verts-in by auto
then show (b, c)  $\in$  SPECTRE G-A
unfolding SPECTRE-def
proof(simp)
have so: Spectre-Order G b c using bcS unfolding SPECTRE-def by auto
then have vv: vote-Spectre G-A app b c = 1
using Spectre-equals-vote-Spectre-honest b-in c-in H1.Honest-Append-One-axioms
by blast
have bbb: ( $\sum i \leftarrow \text{sorted-list-of-set (verts G). vote-Spectre G-A i b c}$ ) =
  ( $\sum i \leftarrow \text{sorted-list-of-set (verts G-A). vote-Spectre G-A i b c}$ ) - vote-Spectre
  G-A app b c
unfolding H1.append-verts-diff

```

```

    using append-diff-sorted-set B1.finite-verts H1.app-in
    by blast
  have sp1: sum-list (map ( $\lambda i.$ 
    (vote-Spectre G-A i b c)) (sorted-list-of-set (verts G-A))) =
    sum-list (map ( $\lambda i.$ 
    (vote-Spectre G-A i b c)) (sorted-list-of-set (verts G))) + 1
    unfolding bbb vv by auto
  have leee: ( $\sum i \leftarrow \text{sorted-list-of-set (verts G). vote-Spectre G i b c}$ )  $\leq$ 
    ( $\sum i \leftarrow \text{sorted-list-of-set (verts G). vote-Spectre G-A i b c}$ )
  proof(rule sum-list-mono)
    fix a
    assume a  $\in$  set (sorted-list-of-set (verts G))
    then have a  $\in$  verts G using sorted-list-of-set(1) H1.finite-verts by auto
    then show vote-Spectre G a b c  $\leq$  vote-Spectre G-A a b c
      using Spectre-Order-Appending-Mono H1.Honest-Append-One-axioms b-in
c-in so by blast
  qed
  have ( $\sum i \leftarrow \text{sorted-list-of-set (verts G). vote-Spectre G i b c}$ )  $\leq$ 
    ( $\sum i \leftarrow \text{sorted-list-of-set (verts G-A). vote-Spectre G-A i b c}$ )
    unfolding sp1 using leee by linarith
  then have signum ( $\sum i \leftarrow \text{sorted-list-of-set (verts G).}$ 
    vote-Spectre G i b c)  $\leq$  signum ( $\sum i \leftarrow \text{sorted-list-of-set (verts G-A).}$ 
    vote-Spectre G-A i b c)
    by(rule signum-mono)
  then have tie-break-int b c (signum ( $\sum i \leftarrow \text{sorted-list-of-set (verts G).}$ 
    vote-Spectre G i b c))
     $\leq$  tie-break-int b c (signum ( $\sum i \leftarrow \text{sorted-list-of-set (verts G-A).}$ 
    vote-Spectre G-A i b c))
    by(rule tie-break-mono)
  then have 1  $\leq$  tie-break-int b c (signum ( $\sum i \leftarrow \text{sorted-list-of-set (verts G-A).}$ 
    vote-Spectre G-A i b c))
    using so
    unfolding Spectre-Order-def by simp
  then show Spectre-Order G-A b c
    unfolding Spectre-Order-def using domain-tie-break by auto
  qed
qed

```

lemma Spectre-equals-vote-Spectre-honest-dishonest:

```

  assumes Append-One-Honest-Dishonest G G-A a G-AB dis
    and b  $\in$  verts G
    and c  $\in$  verts G
  shows Spectre-Order G b c  $\longleftrightarrow$  vote-Spectre G-AB a b c = 1

```

proof –

```

  interpret AOHD: Append-One-Honest-Dishonest G G-A a G-AB dis using
assms(1) by simp
  interpret H2: Append-One G-A G-AB dis using AOHD.app-two by metis
  have b-in: b  $\in$  verts G-A

```

```

    and c-in:  $c \in \text{verts } G\text{-}A$  using AOHD.append-verts-in assms(2,3) by auto
  then have b-inB:  $b \in \text{verts } G\text{-}AB$ 
    and c-inB:  $c \in \text{verts } G\text{-}AB$  using H2.append-verts-in by auto
  have re-all:  $\forall v \in \text{verts } G. a \rightarrow^+_{G\text{-}AB} v$ 
    using AOHD.reachable1-preserve2 assms(2) AOHD.reaches-all AOHD.app-in
    H2.reachable1-preserve
  by simp
  then have r-ab:  $a \rightarrow^+_{G\text{-}AB} b$  using assms(2) by simp
  have r-ac:  $a \rightarrow^+_{G\text{-}AB} c$  using re-all assms(3) by simp
  consider (b-c-eq)  $b = c$  | (not-b-c-eq)  $\neg b = c$  by auto
  then show ?thesis
proof(cases)
  case b-c-eq
  then have Spectre-Order  $G\ b\ c$  using Spectre-Order-reflexive
    AOHD.bD assms by metis
  moreover have vote-Spectre  $G\text{-}AB\ a\ b\ c = 1$ 
    unfolding b-c-eq using vote-Spectre-reflexive
    using H2.bD-A AOHD.app-in2 c-inB by metis
  ultimately show ?thesis by simp
next
  case not-b-c-eq
  then have ee1: vote-Spectre  $G\text{-}AB\ a\ b\ c = (\text{tie-break-int } b\ c\ (\text{signum } (\text{sum-list }
(\text{map } (\lambda i.
(\text{vote-Spectre } (\text{reduce-past } G\text{-}AB\ a)\ i\ b\ c))\ (\text{sorted-list-of-set } (\text{past-nodes } G\text{-}AB\ a))))))
    using vote-Spectre.simps r-ab r-ac
    b-inB c-inB AOHD.app-in2 H2.bD-A
    map-eq-conv
    by simp
  have the-eq: vote-Spectre  $G\text{-}AB\ a\ b\ c = (\text{tie-break-int } b\ c\ (\text{signum } (\text{sum-list }
(\text{map } (\lambda i.
(\text{vote-Spectre } G\ i\ b\ c))\ (\text{sorted-list-of-set } (\text{verts } G))))))
    unfolding AOHD.append-past-nodes2 ee1 AOHD.reduce-append2
    by simp
  show ?thesis
    unfolding the-eq Spectre-Order-def by simp
qed
qed$$ 
```

lemma *Spectre-Order-Appending-Robust:*
assumes *Append-One-Honest-Dishonest* $G\ G\text{-}A\ \text{app } G\text{-}AB\ \text{dis}$
and $a \in \text{verts } G$
and $b \in \text{verts } G$

```

    and  $c \in \text{verts } G$ 
    and Spectre-Order  $G \ b \ c$ 
    shows vote-Spectre  $G \ a \ b \ c \leq \text{vote-Spectre } G\text{-}AB \ a \ b \ c$ 
    using assms
  proof(induction  $G \ a \ b \ c$  rule: vote-Spectre.induct)
    case (1  $G \ a \ b \ c$ )
    interpret AOHD: Append-One-Honest-Dishonest  $G \ G\text{-}A \ \text{app } G\text{-}AB \ \text{dis}$  using 1
  by auto
    interpret HB2: Append-One  $G\text{-}A \ G\text{-}AB \ \text{dis}$  using AOHD.append-two
    by metis
    interpret B2: blockDAG  $G\text{-}A$  using AOHD.bD-A
    by metis
    interpret B3: blockDAG  $G\text{-}AB$  using HB2.bD-A
    by metis
    have a-in-G-A:  $a \in \text{verts } G\text{-}A$ 
    and b-in-G-A:  $b \in \text{verts } G\text{-}A$ 
    and c-in-G-A:  $c \in \text{verts } G\text{-}A$  using 1(4,5,6) AOHD.append-verts-in by auto
    then have a-in-G-AB:  $a \in \text{verts } G\text{-}AB$ 
    and b-in-G-AB:  $b \in \text{verts } G\text{-}AB$ 
    and c-in-G-AB:  $c \in \text{verts } G\text{-}AB$  using 1(4,5,6) HB2.append-verts-in by auto
    show vote-Spectre  $G \ a \ b \ c \leq \text{vote-Spectre } G\text{-}AB \ a \ b \ c$ 
    proof(cases  $a \ b \ c \ G$  rule: Spectre-casesAlt)
      case no-bD
      then show ?thesis using AOHD.bD 1(4,5,6) by auto
    next
      case equal
      then show vote-Spectre  $G \ a \ b \ c \leq \text{vote-Spectre } G\text{-}AB \ a \ b \ c$ 
        using B3.blockDAG-axioms a-in-G-AB b-in-G-AB c-in-G-AB by auto
    next
      case one
      then have  $(a \rightarrow^+_{G\text{-}AB} b \vee a = b) \wedge (\neg a \rightarrow^+_{G\text{-}AB} c)$  using AOHD.reachable1-preserve
        HB2.reachable1-preserve a-in-G-A b-in-G-A c-in-G-A by auto
      then show ?thesis using one B3.blockDAG-axioms a-in-G-AB b-in-G-AB
        c-in-G-AB by auto
    next
      case two
      then have  $\neg((a \rightarrow^+_{G\text{-}AB} b \vee a = b) \wedge (\neg a \rightarrow^+_{G\text{-}AB} c))$ 
        using AOHD.reachable1-preserve
        HB2.reachable1-preserve a-in-G-A b-in-G-A c-in-G-A by auto
      moreover have  $(a \rightarrow^+_{G\text{-}AB} c \vee a = c) \wedge (\neg a \rightarrow^+_{G\text{-}AB} b)$ 
        using two AOHD.reachable1-preserve
        HB2.reachable1-preserve a-in-G-A b-in-G-A c-in-G-A by auto
      ultimately show ?thesis using two by auto
    next
      have past-nodes  $G\text{-}AB \ a = \text{past-nodes } G\text{-}A \ a$  using a-in-G-A HB2.append-past-nodes
    by auto
    then have ppp: past-nodes  $G\text{-}AB \ a = \text{past-nodes } G \ a$  using 1(4) AOHD.append-past-nodes
    by auto
    have reduce-past  $G\text{-}AB \ a = \text{reduce-past } G\text{-}A \ a$  using a-in-G-A HB2.append-reduce-some

```

```

by auto
  then have rrp: reduce-past G-AB a = reduce-past G a using 1(4) AOHD.append-reduce-some
by auto
  case three
    then have ins: vote-Spectre G a b c = (tie-break-int b c (signum (sum-list
(map (λi.
  (vote-Spectre (reduce-past G a) i b c)) (sorted-list-of-set (past-nodes G a))))))
    by auto
    have bneqc: b ≠ c using three by simp
    have ¬((a →+G-AB b ∨ a = b) ∧ (¬ a →+G-AB c))
      using three AOHD.reachable1-preserve HB2.reachable1-preserve a-in-G-A
b-in-G-A c-in-G-A by auto
    moreover have ¬((a →+G-AB c ∨ a = c) ∧ (¬ a →+G-AB b))
      using three AOHD.reachable1-preserve HB2.reachable1-preserve a-in-G-A
b-in-G-A c-in-G-A by auto
    moreover have a →+G-AB b ∧ a →+G-AB c
      using three AOHD.reachable1-preserve HB2.reachable1-preserve a-in-G-A
b-in-G-A c-in-G-A by auto
    ultimately have vote-Spectre G-AB a b c = (tie-break-int b c (signum
(sum-list (map (λi.
  (vote-Spectre (reduce-past G-AB a) i b c)) (sorted-list-of-set (past-nodes G-AB
a))))))
    using a-in-G-AB b-in-G-AB c-in-G-AB B3.blockDAG-axioms vote-Spectre.simps
bneqc by auto
    then have ins-2: vote-Spectre G-AB a b c = (tie-break-int b c (signum (sum-list
(map (λi.
  (vote-Spectre (reduce-past G a) i b c)) (sorted-list-of-set (past-nodes G a))))))
    unfolding ppp rrp by auto
    show ?thesis unfolding ins ins-2 by simp
next
  case four
    then have ins: vote-Spectre G a b c = signum (sum-list (map (λi.
(vote-Spectre G i b c)) (sorted-list-of-set (future-nodes G a))))
    by auto
    have bneqc: b ≠ c using four by simp
    moreover have ¬((a →+G-AB b ∨ a = b) ∧ (¬ a →+G-AB c))
      using four AOHD.reachable1-preserve HB2.reachable1-preserve a-in-G-A
b-in-G-A c-in-G-A by auto
    moreover have ¬((a →+G-AB c ∨ a = c) ∧ (¬ a →+G-AB b))
      using four AOHD.reachable1-preserve HB2.reachable1-preserve a-in-G-A
b-in-G-A c-in-G-A by auto
    moreover have ¬(a →+G-AB b ∧ a →+G-AB c) using four AOHD.reachable1-preserve
      using four AOHD.reachable1-preserve HB2.reachable1-preserve a-in-G-A
b-in-G-A c-in-G-A by auto
    ultimately have ins2:
      vote-Spectre G-AB a b c = signum (sum-list (map (λi.
(vote-Spectre G-AB i b c)) (sorted-list-of-set (future-nodes G-AB a))))
    using B3.blockDAG-axioms a-in-G-AB b-in-G-AB c-in-G-AB vote-Spectre.simps

```

```

four by auto
  have  $\bigwedge x . x \in \text{set } (\text{sorted-list-of-set } (\text{future-nodes } G \ a)) \implies x \in \text{verts } G$ 
   $\implies \text{vote-Spectre } G \ x \ b \ c \leq \text{vote-Spectre } G\text{-AB } x \ b \ c$ 
  using 1(2,3) four
  1.premis(5) by blast
  then have fut:  $\forall x \in \text{set } (\text{sorted-list-of-set } (\text{future-nodes } G \ a)).$ 
     $(\text{vote-Spectre } G \ x \ b \ c) \leq (\text{vote-Spectre } G\text{-AB } x \ b \ c)$ 
  using AOHD.finite-past AOHD.past-nodes-verts
  by simp
  consider (disnin)  $\text{dis} \notin \text{future-nodes } G\text{-AB } a \mid (\text{disin}) \text{dis} \in \text{future-nodes } G\text{-AB}$ 
a by auto
  then show  $\text{vote-Spectre } G \ a \ b \ c \leq \text{vote-Spectre } G\text{-AB } a \ b \ c$ 
  using 1(4)
  proof(cases)
    case disnin
    then have futN:  $\text{future-nodes } G\text{-AB } a - \{\text{app}\} = \text{future-nodes } G \ a$ 
    using AOHD.append-future 1(4) HB2.append-future a-in-G-A
    by (metis Diff-idemp Diff-insert-absorb)
    then have appfut:  $\text{app} \in \text{future-nodes } G\text{-AB } a$  using AOHD.app-in-future2
    1(4) by auto
    then have bbb:  $\text{sum-list } (\text{map } (\lambda i.$ 
       $(\text{vote-Spectre } G\text{-AB } i \ b \ c)) (\text{sorted-list-of-set } (\text{future-nodes } G \ a)))$ 
    =  $\text{sum-list } (\text{map } (\lambda i.$ 
       $(\text{vote-Spectre } G\text{-AB } i \ b \ c)) (\text{sorted-list-of-set } (\text{future-nodes } G\text{-AB } a)))$ 
    -  $(\text{vote-Spectre } G\text{-AB } \text{app} \ b \ c)$ 
    using futN append-diff-sorted-set B3.finite-future by metis
    have  $\text{vote-Spectre } G\text{-AB } \text{app} \ b \ c = 1$ 
    using 1.premis Spectre-equals-vote-Spectre-honest-dishonest
    by blast
    then have sp1:  $\text{sum-list } (\text{map } (\lambda i.$ 
       $(\text{vote-Spectre } G\text{-AB } i \ b \ c)) (\text{sorted-list-of-set } (\text{future-nodes } G\text{-AB } a))) =$ 
     $\text{sum-list } (\text{map } (\lambda i.$ 
       $(\text{vote-Spectre } G\text{-AB } i \ b \ c)) (\text{sorted-list-of-set } (\text{future-nodes } G \ a))) + 1$ 
    using bbb by auto
    have sp2:  $\text{sum-list } (\text{map } (\lambda i.(\text{vote-Spectre } G \ i \ b \ c))$ 
       $(\text{sorted-list-of-set } (\text{future-nodes } G \ a)))$ 
       $\leq \text{sum-list } (\text{map } (\lambda i.$ 
       $(\text{vote-Spectre } G\text{-AB } i \ b \ c)) (\text{sorted-list-of-set } (\text{future-nodes } G \ a)))$ 
    by (metis fut sum-list-mono)
    show ?thesis unfolding ins ins2
    proof (rule signum-mono)
      show  $(\sum i \leftarrow \text{sorted-list-of-set } (\text{future-nodes } G \ a). \text{vote-Spectre } G \ i \ b \ c)$ 
       $\leq (\sum i \leftarrow \text{sorted-list-of-set } (\text{future-nodes } G\text{-AB } a). \text{vote-Spectre } G\text{-AB } i \ b \ c)$ 
      unfolding sp1 using sp2
      by linarith
    qed
  next
  case disin
  have futN:  $\text{future-nodes } G\text{-AB } a - \{\text{app}\} - \{\text{dis}\} = \text{future-nodes } G \ a$ 

```

```

    using AOHD.append-future 1(4) HB2.append-future a-in-G-A
  by auto
then have appfut: app ∈ future-nodes G-AB a using AOHD.app-in-future2
  1(4) by auto
then have bbb: sum-list (map (λi.
  (vote-Spectre G-AB i b c)) (sorted-list-of-set (future-nodes G a)))
= sum-list (map (λi.
  (vote-Spectre G-AB i b c)) (sorted-list-of-set (future-nodes G-AB a)))
- (vote-Spectre G-AB app b c) - (vote-Spectre G-AB dis b c)
  using futN disin append-diff-sorted-set B3.finite-future
  by (metis (no-types, lifting) AOHD.app-not-dis finite-Diff insert-Diff-single
    insert-absorb2 insert-iff mk-disjoint-insert)
have v1: vote-Spectre G-AB app b c = 1
  using 1.premis Spectre-equals-vote-Spectre-honest-dishonest
  by blast
have sp1: sum-list (map (λi.
  (vote-Spectre G-AB i b c)) (sorted-list-of-set (future-nodes G-AB a))) =
  sum-list (map (λi.
  (vote-Spectre G-AB i b c)) (sorted-list-of-set (future-nodes G a))) + 1
  + (vote-Spectre G-AB dis b c)
  unfolding bbb v1 by auto
have sp2: sum-list (map (λi.(vote-Spectre G i b c))
  (sorted-list-of-set (future-nodes G a)))
  ≤ sum-list (map (λi.
  (vote-Spectre G-AB i b c)) (sorted-list-of-set (future-nodes G a)))
  by (metis fut sum-list-mono)
show ?thesis unfolding ins ins2
proof (rule signum-mono)
  consider vote-Spectre G-AB dis b c = 1 | vote-Spectre G-AB dis b c = 0
  | vote-Spectre G-AB dis b c = -1 using domain-Spectre
  by blast
  then show (∑ i←sorted-list-of-set (future-nodes G a). vote-Spectre G i b c)
  ≤ (∑ i←sorted-list-of-set (future-nodes G-AB a). vote-Spectre G-AB i b c)
    unfolding sp1 using sp2 proof(cases, auto)
    qed
  qed
qed
qed
qed
qed

```

lemma *One-Appending-Robust SPECTRE*

unfolding *One-Appending-Robust-def*

proof *safe*

fix G $G-A$ $G-AB::('a::linorder, 'b)$ *pre-digraph* **and** $app\ b\ c\ dis::'a$

assume *Append-One-Honest-Dishonest* $G\ G-A\ app\ G-AB\ dis$

and $bcS::(b, c) \in SPECTRE\ G$

then interpret $H1::Append-One-Honest-Dishonest\ G\ G-A\ app\ G-AB\ dis$ **by**
auto

interpret $H2::Append-One\ G-A\ G-AB\ dis$ **using** $H1.app-two$ **by** *auto*

have $b\text{-in}$: $b \in \text{verts } G$
and $c\text{-in}$: $c \in \text{verts } G$ **using** bcS **unfolding** $SPECTRE\text{-def}$ **by** *auto*
then have $b\text{-in}2$: $b \in \text{verts } G\text{-}A$
and $c\text{-in}2$: $c \in \text{verts } G\text{-}A$ **using** $H1.\text{append-verts-in}$ **by** *auto*
then have $b\text{-in}3$: $b \in \text{verts } G\text{-}AB$
and $c\text{-in}3$: $c \in \text{verts } G\text{-}AB$ **using** $H2.\text{append-verts-in}$ **by** *auto*
then show $(b, c) \in SPECTRE\ G\text{-}AB$
unfolding $SPECTRE\text{-def}$
proof(*simp*)
have so : $Spectre\text{-Order } G\ b\ c$ **using** bcS **unfolding** $SPECTRE\text{-def}$ **by** *auto*
then have vv : $\text{vote-Spectre } G\text{-}AB\ \text{app } b\ c = 1$
using $Spectre\text{-equals-vote-Spectre-honest-dishonest } b\text{-in } c\text{-in}$
 $H1.\text{Append-One-Honest-Dishonest-axioms}$
by *blast*
have fff : $\text{finite } (\text{verts } G\text{-}AB)$ **using** $H2.bD\text{-}A\ \text{fin-digraph.finite-verts subs}$ **by** *auto*
then have bbb : $(\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G). \text{vote-Spectre } G\text{-}AB\ i\ b\ c) =$
 $(\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G\text{-}AB). \text{vote-Spectre } G\text{-}AB\ i\ b\ c) - \text{vote-Spectre}$
 $G\text{-}AB\ \text{dis } b\ c$
 $- \text{vote-Spectre } G\text{-}AB\ \text{app } b\ c$
unfolding $H1.\text{append-verts-diff } H2.\text{append-verts-diff}$
using $H1.\text{app-not-dis } H1.\text{app-in}2\ H2.\text{app-in append-diff-sorted-set2}$
by *blast*
have $sp1$: $\text{sum-list } (\text{map } (\lambda i.$
 $(\text{vote-Spectre } G\text{-}AB\ i\ b\ c))\ (\text{sorted-list-of-set } (\text{verts } G\text{-}AB))) =$
 $\text{sum-list } (\text{map } (\lambda i.$
 $(\text{vote-Spectre } G\text{-}AB\ i\ b\ c))\ (\text{sorted-list-of-set } (\text{verts } G))) + 1 + \text{vote-Spectre}$
 $G\text{-}AB\ \text{dis } b\ c$
unfolding $bbb\ vv$ **by** *auto*
have lee : $(\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G). \text{vote-Spectre } G\ i\ b\ c) \leq$
 $(\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G). \text{vote-Spectre } G\text{-}AB\ i\ b\ c)$
proof(*rule sum-list-mono*)
fix a
assume $a \in \text{set } (\text{sorted-list-of-set } (\text{verts } G))$
then have $a \in \text{verts } G$ **using** $\text{sorted-list-of-set}(1)\ H1.\text{finite-verts}$ **by** *auto*
then show $\text{vote-Spectre } G\ a\ b\ c \leq \text{vote-Spectre } G\text{-}AB\ a\ b\ c$
using $Spectre\text{-Order-Appending-Robust } H1.\text{Append-One-Honest-Dishonest-axioms}$
 $b\text{-in } c\text{-in so}$
by *blast*
qed
consider $\text{vote-Spectre } G\text{-}AB\ \text{dis } b\ c = 1 \mid \text{vote-Spectre } G\text{-}AB\ \text{dis } b\ c = 0$
 $\mid \text{vote-Spectre } G\text{-}AB\ \text{dis } b\ c = -1$ **using** domain-Spectre
by *blast*
then have $(\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G). \text{vote-Spectre } G\ i\ b\ c) \leq$
 $(\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G\text{-}AB). \text{vote-Spectre } G\text{-}AB\ i\ b\ c)$
unfolding $sp1$ **using** lee **proof**(*cases, auto*) **qed**
then have $\text{signum } (\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G).$
 $\text{vote-Spectre } G\ i\ b\ c) \leq \text{signum } (\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G\text{-}AB).$
 $\text{vote-Spectre } G\text{-}AB\ i\ b\ c)$


```

    by(rule signum-mono)
  then have tie-break-int b c (signum ( $\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G).$ 
    vote-Spectre G i b c))
     $\leq$  tie-break-int b c (signum ( $\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G-AB).$ 
    vote-Spectre G-AB i b c))
  by(rule tie-break-mono)
  then have  $1 \leq$  tie-break-int b c (signum ( $\sum i \leftarrow \text{sorted-list-of-set } (\text{verts } G-AB).$ 
    vote-Spectre G-AB i b c))
  using so
  unfolding Spectre-Order-def by simp
  then show Spectre-Order G-AB b c
  unfolding Spectre-Order-def using domain-tie-break by auto
qed
qed

end

```

```

theory Codegen
  imports blockDAG Spectre Ghostdag Extend-blockDAG
begin

```

8 Code Generation

```

fun arcAlt:: ('a,'b) pre-digraph  $\Rightarrow$  'b  $\Rightarrow$  'a  $\times$  'a  $\Rightarrow$  bool
  where arcAlt G e uv = (e  $\in$  arcs G  $\wedge$  tail G e = fst uv  $\wedge$  head G e = snd uv)

fun iterate:: ('a  $\Rightarrow$  bool)  $\Rightarrow$  'a set  $\Rightarrow$  bool
  where iterate S P = Finite-Set.fold ( $\lambda r A. S r \wedge A$ ) False P

lemma (in DAG) arcAlt-eq:
  shows arcAlt G e uv = wf-digraph.arc G e uv
  unfolding arc-def arcAlt.simps by simp

lemma [code]: blockDAG G = (DAG G  $\wedge$  (( $\exists p \in \text{verts } G. ((\forall r \in \text{verts } G. (r \rightarrow^+_G p \vee r = p)))$ )  $\wedge$ 
  ( $\forall e \in (\text{arcs } G). \forall u \in \text{verts } G. \forall v \in \text{verts } G.$ 
  ( $u \rightarrow^+ (\text{pre-digraph.del-arc } G \ e) \ v$ )  $\longrightarrow \neg \text{arcAlt } G \ e \ (u,v)$ )))
  using DAG.arcAlt-eq wf-digraph-def DAG.axioms(1)
  digraph.axioms(1) fin-digraph.axioms(1) wf-digraph.arcE blockDAG-axioms-def
  blockDAG-def
  by metis

lemma [code]: DAG G = (digraph G  $\wedge$  ( $\forall v \in \text{verts } G. \neg(v \rightarrow^+_G v)$ ))
  unfolding DAG-axioms-def DAG-def
  by (metis digraph.axioms(1) fin-digraph.axioms(1) wf-digraph.reachable1-in-verts(1))

```

```

lemma [code]: digraph  $G = (\text{fin-digraph } G \wedge \text{loopfree-digraph } G \wedge \text{nomulti-digraph } G)$ 
  unfolding digraph-def by auto

lemma [code]: wf-digraph  $G = ($ 
   $(\forall e \in \text{arcs } G. \text{tail } G \ e \in \text{verts } G) \wedge$ 
   $(\forall e \in \text{arcs } G. \text{head } G \ e \in \text{verts } G))$ 
  using wf-digraph-def by auto

lemma [code]: nomulti-digraph  $G = (\text{wf-digraph } G \wedge$ 
   $(\forall e1 \in \text{arcs } G. \forall e2 \in \text{arcs } G .$ 
     $\text{arc-to-ends } G \ e1 = \text{arc-to-ends } G \ e2 \longrightarrow e1 = e2))$ 
  unfolding nomulti-digraph-def nomulti-digraph-axioms-def by auto

lemma [code]: loopfree-digraph  $G = (\text{wf-digraph } G \wedge (\forall e \in \text{arcs } G. \text{tail } G \ e \neq$ 
   $\text{head } G \ e))$ 
  unfolding loopfree-digraph-def loopfree-digraph-axioms-def by auto

lemma [code]: pre-digraph.del-arc  $G \ a =$ 
   $(\text{verts} = \text{verts } G, \text{arcs} = \text{arcs } G - \{a\}, \text{tail} = \text{tail } G, \text{head} = \text{head } G)$ 
  by (simp add: pre-digraph.del-arc-def)

lemma [code]: fin-digraph  $G = (\text{wf-digraph } G \wedge (\text{card } (\text{verts } G) > 0 \vee \text{verts } G =$ 
   $\{\}))$ 
   $\wedge ((\text{card } (\text{arcs } G) > 0 \vee \text{arcs } G = \{\}))$ 
  using card-ge-0-finite fin-digraph-def fin-digraph-axioms-def
  by (metis card-gt-0-iff finite.emptyI)

fun vote-Spectre-Int:: (integer, integer×integer) pre-digraph  $\Rightarrow$ 
  integer  $\Rightarrow$  integer  $\Rightarrow$  integer  $\Rightarrow$  integer
  where vote-Spectre-Int  $V \ a \ b \ c = \text{integer-of-int } (\text{vote-Spectre } V \ a \ b \ c)$ 

fun SpectreOrder-Int:: (integer, integer×integer) pre-digraph  $\Rightarrow$  integer  $\Rightarrow$  integer
   $\Rightarrow$  bool
  where SpectreOrder-Int  $G = \text{Spectre-Order } G$ 

fun OrderDAG-Int:: (integer, integer×integer) pre-digraph  $\Rightarrow$ 
  integer  $\Rightarrow$  (integer set × integer list)
  where OrderDAG-Int  $V \ a = (\text{OrderDAG } V \ (\text{nat-of-integer } a))$ 

export-code top-sort anticone set blockDAG pre-digraph-ext snd fst vote-Spectre-Int
  SpectreOrder-Int OrderDAG-Int
in Haskell module-name DAGS file code/

notepad begin

```

```

let ?G = (verts = {1::int,2,3,4,5,6,7,8,9,10}, arcs = {(2,1),(3,1),(4,1),
(5,2),(6,3),(7,4),(8,5),(8,3),(9,6),(9,4),(10,7),(10,2)}, tail = fst, head = snd)
let ?a = 2
let ?b = 3
let ?c = 4
value blockDAG ?G
value Spectre-Order ?G ?a ?b  $\wedge$  Spectre-Order ?G ?b ?c  $\wedge \neg$  Spectre-Order ?G
?a ?c
end

```

8.1 Extend Graph

```

declare pre-digraph.del-vert-def [code]
declare Append-One-axioms-def [code]
declare Honest-Append-One-axioms-def [code]
declare Append-One-def [code]
declare Honest-Append-One-def [code]
declare Append-One-Honest-Dishonest-axioms-def [code]
declare Append-One-Honest-Dishonest-def [code]

```

```

end
theory Ghostdag-Properties
  imports Ghostdag Extend-blockDAG Properties Codegen
begin

```

9 GHOSTDAG properties

9.1 GHOSTDAG Order Preserving

```

lemma GhostDAG-preserving:
  assumes blockDAG G
  and  $x \rightarrow^+_G y$ 
  shows  $(y,x) \in \text{GHOSTDAG } k \ G$ 
  unfolding GHOSTDAG.simps using assms
proof(induct G k arbitrary: x y rule: OrderDAG.induct )
  case (1 G k)
  then show ?case proof (cases G rule: OrderDAG-casesAlt)
    case ntB
    then show ?thesis using 1 by auto
  next
  case one
  then have  $\neg x \rightarrow^+_G y$ 
  using subs wf-digraph.reachable1-in-verts 1
  by (metis DAG.cycle-free OrderDAG-casesAlt blockDAG.reduce-less
    blockDAG.reduce-past-dagbased blockDAG.unique-genesis less-one not-one-less-zero)

```

```

    then show ?thesis using 1 by simp
next
case more
obtain pp where pp-in: pp = (map (λi. (OrderDAG (reduce-past G i) k, i))
  (sorted-list-of-set (tips G))) using blockDAG.tips-exist by auto
have backw: list-to-rel (snd (OrderDAG G k)) =
  list-to-rel (snd (fold (app-if-blue-else-add-end G k)
    (top-sort G (sorted-list-of-set (anticone G (snd (choose-max-blue-set
pp))))))
  (add-set-list-tuple (choose-max-blue-set pp))))
  using OrderDAG.simps less-irrefl-nat more pp-in
  by (metis (mono-tags, lifting))
obtain S where s-in:
  (top-sort G (sorted-list-of-set (anticone G (snd (choose-max-blue-set pp))))))
= S by simp
obtain t where t-in : (add-set-list-tuple (choose-max-blue-set pp)) = t by simp
obtain ma where ma-def: ma = (snd (choose-max-blue-set pp)) by simp
have ma-vert: ma ∈ verts G unfolding ma-def using chosen-map-simps(2)
digraph.tips-in-verts
  more(1) subs subsetD pp-in by blast
have ma-tip: is-tip G ma unfolding ma-def
  using chosen-map-simps(2) more pp-in tips-tips
  by (metis (no-types))
then have no-gen: ¬ blockDAG.is-genesis-node G ma unfolding ma-def using
pp-in
  blockDAG.tips-unequal-gen more
  by metis
then have red-bd: blockDAG (reduce-past G ma)
  using blockDAG.reduce-past-dagbased more ma-vert unfolding ma-def
  by auto
consider (ind) x ∈ past-nodes G ma ∧ y ∈ past-nodes G ma
  |(x-in) x ∉ past-nodes G ma ∧ y ∈ past-nodes G ma
  |(y-in) x ∈ past-nodes G ma ∧ y ∉ past-nodes G ma
  |(both-nin) x ∉ past-nodes G ma ∧ y ∉ past-nodes G ma by auto
then show ?thesis proof(cases)
case ind
then have x →+ reduce-past G ma y using DAG.reduce-past-path2 more
  1 subs
  by (metis)
moreover have ma-tips: ma ∈ set (sorted-list-of-set (tips G))
  using chosen-map-simps(1) pp-in more(1)
  unfolding ma-def by auto
ultimately have (y,x) ∈ list-to-rel (snd (OrderDAG (reduce-past G ma) k))
  unfolding ma-def
  using more 1 ind less-numeral-extra(4) ma-def red-bd
  by (metis)
then have (y,x) ∈ list-to-rel (snd (fst (choose-max-blue-set pp)))
  using chosen-map-simps(6) pp-in 1 unfolding ma-def by fastforce
then have rel-base: (y,x) ∈ list-to-rel (snd (add-set-list-tuple(choose-max-blue-set

```

```

pp)))
  using add-set-list-tuple.simps list-to-rel-mono prod.collapse snd-conv
  by metis

show ?thesis
  unfolding ma-def backw s-in
  using rel-base unfolding t-in
  using fold-app-mono-rel prod.collapse
  by metis
next
case x-in
then have  $y \in \text{set} (\text{snd} (\text{OrderDAG} (\text{reduce-past } G \text{ ma}) k))$ 
unfolding reduce-past.simps using induce-subgraph-verts Verts-in-OrderDAG

  more red-bd reduce-past.elims
  by (metis)
then have  $y\text{-in-base: } y \in \text{set} (\text{snd} (\text{fst} (\text{choose-max-blue-set } pp)))$ 
unfolding ma-def using chosen-map-simps(6) more pp-in
by fastforce
consider  $(x-t) \ x = \text{ma} \mid (x\text{-ant}) \ x \in \text{anticone } G \text{ ma}$  using DAG.verts-comp2
  subs 1 ma-tip ma-vert
  mem-Collect-eq tips-def wf-digraph.reachable1-in-verts(1) x-in
  by (metis (no-types, lifting))
then show ?thesis proof(cases)
  case x-t
  then have  $(y,x) \in \text{list-to-rel} (\text{snd} (\text{add-set-list-tuple} (\text{choose-max-blue-set } pp)))$ 
  unfolding x-t ma-def
  using y-in-base add-set-list-tuple.simps list-to-rel-append prod.collapse sndI
  by metis
  then show ?thesis unfolding ma-def backw s-in
  unfolding t-in
  using fold-app-mono-rel prod.collapse
  by metis
next
case x-ant
then have  $x \in \text{set} (\text{sorted-list-of-set} (\text{anticone } G \text{ ma}))$ 
  using sorted-list-of-set(1) more subs
  by (metis DAG.anticon-finite)
moreover have  $y \in \text{set} (\text{snd} (\text{add-set-list-tuple} (\text{choose-max-blue-set } pp)))$ 
  using add-set-list-tuple-mono in-mono prod.collapse y-in-base
  by (metis (mono-tags, lifting))
ultimately show ?thesis unfolding backw
  by (metis fold-app-app-rel ma-def prod.collapse top-sort-con)
qed
next
case y-in
then have  $y \in \text{past-nodes } G \text{ ma}$  unfolding past-nodes.simps using 1(2,3)
  wf-digraph.reachable1-in-verts(2) subs mem-Collect-eq trancl-trans

```

```

    by (metis (mono-tags, lifting))
  then show ?thesis using y-in by simp
next
case both-nin
consider (x-t) x = ma | (x-ant) x ∈ anticone G ma using DAG.verts-comp2
  subs 1 ma-tip ma-vert
  mem-Collect-eq tips-def wf-digraph.reachable1-in-verts(1) both-nin
  by (metis (no-types, lifting))
then show ?thesis proof(cases)
case x-t
have y ∈ past-nodes G ma using 1(3) more
  past-nodes.simps unfolding x-t
  by (simp add: subs wf-digraph.reachable1-in-verts(2))
then show ?thesis using both-nin by simp
next
have y-ina: y ∈ anticone G ma
proof(rule ccontr)
  assume ¬ y ∈ anticone G ma
  then have y = ma
    unfolding anticone.simps using subs wf-digraph.reachable1-in-verts(2)
  1(2,3)
    ma-tip both-nin
    by fastforce
  then have x →+G ma using 1(3) by auto
  then show False using subs 1(2)
    by (metis wf-digraph.tips-not-referenced ma-tip)
qed
case x-ant
then have (y,x) ∈ list-to-rel (top-sort G (sorted-list-of-set (anticone G ma)))
  using y-ina DAG.anticon-finite subs 1(2,3) sorted-list-of-set(1) top-sort-rel
  by metis
then show ?thesis unfolding backw ma-def using
  fold-app-mono list-to-rel-mono2
  by (metis old.prod.exhaust)
qed
qed
qed
qed

```

```

lemma ∀ k. Order-Preserving (GHOSTDAG k)
  unfolding Order-Preserving-def
  using GhostDAG-preserving
  by blast

```

9.2 GHOSTDAG Linear Order

```

lemma GhostDAG-linear:
  assumes blockDAG G

```

shows *linear-order-on* (verts G) (*GHOSTDAG* k G)
unfolding *GHOSTDAG.simps*
using *list-order-linear OrderDAG-distinct OrderDAG-total assms* **by** *metis*

lemma $\forall k$. *Linear-Order* (*GHOSTDAG* k)
unfolding *Linear-Order-def*
using *GhostDAG-linear* **by** *blast*

9.3 GHOSTDAG One Appending Monotone

lemma *OrderDAG-append-one*:
assumes *Honest-Append-One* G $G-A$ a
shows *snd* (*OrderDAG* $G-A$ k) = *snd* (*OrderDAG* G k) @ $[a]$
proof –
have $bD-A$: *blockDAG* $G-A$ **using** *assms Append-One.bD-A Honest-Append-One-def*
by *metis*
have $g1$: *card* (verts $G-A$) $\neq 1$
using *assms Append-One.append-greater-1 Honest-Append-One-def less-not-refl*
by *metis*
have (*tips* $G-A$) = $\{a\}$ **using** *Honest-Append-One.append-is-only-tip assms* **by** *metis*
then have *tips-app*: (*sorted-list-of-set* (*tips* $G-A$)) = $[a]$ **by** *auto*
obtain *the-map* **where** *the-map-in*:
the-map = ((*map* (λi .(((*OrderDAG* (*reduce-past* $G-A$ i) k)) , i)) (*sorted-list-of-set* (*tips* $G-A$))))
by *auto*
then have $m-l$: *the-map* = [(((*OrderDAG* (*reduce-past* $G-A$ a) k), a)]
unfolding *the-map-in* **using** *tips-app* **by** *auto*
then have $c-l$: *choose-max-blue-set* *the-map*
= ((*OrderDAG* (*reduce-past* $G-A$ a) k), a)
by (*metis* (*no-types*, *lifting*) *choose-max-blue-avoid-empty list.discI list.set-cases set-ConsD*)
then have bb : *choose-max-blue-set* *the-map*
= ((*OrderDAG* G k), a) **using** *Honest-Append-One.reduce-append assms*
by *metis*
let $?M$ = *choose-max-blue-set* *the-map*
have *anticon* $G-A$ (*snd* $?M$) = $\{\}$
unfolding $c-l$
using *assms Honest-Append-One.append-no-anticon* *sndI*
by *metis*
then have eml : (*top-sort* $G-A$ (*sorted-list-of-set* (*anticon* $G-A$ (*snd* $?M$)))) = $[]$
by (*metis* *sorted-list-of-set-empty top-sort.simps(1)*)
then have (*fold* (*app-if-blue-else-add-end* G k)
(*top-sort* $G-A$ (*sorted-list-of-set* (*anticon* $G-A$ (*snd* $?M$))))
(*add-set-list-tuple* $?M$)) = (*add-set-list-tuple* $?M$)
using bb **by** *simp*
moreover have *snd* (*add-set-list-tuple* $?M$) = *snd* (*OrderDAG* G k) @ $[a]$
unfolding bb
using *add-set-list-tuple.simps Pair-inject add-set-list-tuple.elims snd-conv*

```

    by (metis (mono-tags, lifting))
  ultimately show ?thesis
    unfolding the-map-in
    using OrderDAG.simps bD-A g1 eml fold-simps(1) list.simps(8) list.simps(9)
  the-map-in tips-app
    by (metis (no-types, lifting))
qed

```

```

lemma  $\forall k$ . One-Appending-Monotone (GHOSTDAG k)
  unfolding One-Appending-Monotone-def GHOSTDAG.simps
  using list-to-rel-mono OrderDAG-append-one
  by metis

```

9.4 GHOSTDAG Not One Appending Robust

```

datatype FV = V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | V10

```

```

fun FV-Suc :: FV  $\Rightarrow$  FV set
  where
    FV-Suc V1 = { V1, V2, V3, V4, V5, V6, V7, V8, V9, V10 } |
    FV-Suc V2 = { V2, V3, V4, V5, V6, V7, V8, V9, V10 } |
    FV-Suc V3 = { V3, V4, V5, V6, V7, V8, V9, V10 } |
    FV-Suc V4 = { V4, V5, V6, V7, V8, V9, V10 } |
    FV-Suc V5 = { V5, V6, V7, V8, V9, V10 } |
    FV-Suc V6 = { V6, V7, V8, V9, V10 } |
    FV-Suc V7 = { V7, V8, V9, V10 } |
    FV-Suc V8 = { V8, V9, V10 } |
    FV-Suc V9 = { V9, V10 } |
    FV-Suc V10 = { V10 }

```

```

fun less-eq-FV :: FV  $\Rightarrow$  FV  $\Rightarrow$  bool
  where less-eq-FV a b = (b  $\in$  FV-Suc a)

```

```

fun less-FV :: FV  $\Rightarrow$  FV  $\Rightarrow$  bool
  where less-FV a b = (a  $\neq$  b  $\wedge$  less-eq-FV a b)

```

```

lemma FV-cases:
  fixes x::FV
  obtains x = V1 | x = V2 | x = V3 | x = V4 | x = V5 | x = V6 | x = V7 | x
    = V8
    | x = V9 | x = V10
  proof(cases x, auto) qed

```

```

instantiation FV :: linorder
begin
  definition less-eq  $\equiv$  less-eq-FV
  definition less  $\equiv$  less-FV

```

```

instance
proof(standard)

```



```

fix x y z :: FV
show  $x \leq x$  unfolding less-eq-FV-def less-eq-FV.simps
proof(cases x, auto) qed
show  $x \leq y \vee y \leq x$ 
  unfolding less-eq-FV-def less-eq-FV.simps
  by(cases x rule: FV-cases) (cases y rule: FV-cases, auto)+
show  $(x < y) = (x \leq y \wedge \neg y \leq x)$ 
unfolding less-FV-def less-FV.simps less-eq-FV-def less-eq-FV.simps
by(cases x rule: FV-cases) (cases y rule: FV-cases, auto)+
show  $x \leq y \implies y \leq x \implies x = y$ 
unfolding less-FV-def less-FV.simps less-eq-FV-def less-eq-FV.simps
by (cases x rule: FV-cases)(cases y rule: FV-cases, auto)+
show  $x \leq y \implies y \leq z \implies x \leq z$ 
unfolding less-FV-def less-FV.simps less-eq-FV-def less-eq-FV.simps
by (cases x rule: FV-cases)(cases y rule: FV-cases, auto)+
qed
end

instantiation FV :: enum
begin

definition enum-FV  $\equiv [V1, V2, V3, V4, V5, V6, V7, V8, V9, V10]$ 

fun enum-all-FV:: (FV  $\Rightarrow$  bool)  $\Rightarrow$  bool
where enum-all-FV P = Ball { V1, V2, V3, V4, V5, V6, V7, V8, V9, V10} P

fun enum-ex-FV:: (FV  $\Rightarrow$  bool)  $\Rightarrow$  bool
where enum-ex-FV P = Bex { V1, V2, V3, V4, V5, V6, V7, V8, V9, V10} P

instance
  apply(standard)
  apply(simp-all)
  unfolding enum-FV-def UNIV-def
proof –
  show {x. True} = set [V1, V2, V3, V4, V5, V6, V7, V8, V9, V10]
  proof safe
    fix x::FV
    show  $x \in \text{set } [V1, V2, V3, V4, V5, V6, V7, V8, V9, V10]$ 
    using FV-cases by auto
  qed
  show distinct [V1, V2, V3, V4, V5, V6, V7, V8, V9, V10] by auto
  fix P
  show A: (P V1  $\wedge$  P V2  $\wedge$  P V3  $\wedge$  P V4  $\wedge$  P V5  $\wedge$  P V6  $\wedge$  P V7  $\wedge$  P V8  $\wedge$  P
V9  $\wedge$  P V10) = All P
    unfolding All-def
  proof(standard, auto, standard)
    fix x
    show P V1  $\implies$ 
      P V2  $\implies$  P V3  $\implies$  P V4  $\implies$  P V5  $\implies$  P V6  $\implies$  P V7  $\implies$ 

```

```

    P V8  $\implies$  P V9  $\implies$  P V10  $\implies$  P x = True
    proof(cases x rule: FV-cases, auto) qed
  qed
  show (P V1  $\vee$  P V2  $\vee$  P V3  $\vee$  P V4  $\vee$  P V5  $\vee$  P V6  $\vee$  P V7  $\vee$  P V8  $\vee$  P V9
 $\vee$  P V10) = Ex P
  proof(safe, auto)
    fix x::FV
    show P x  $\implies$ 
       $\neg$  P V1  $\implies$ 
       $\neg$  P V2  $\implies$   $\neg$  P V3  $\implies$   $\neg$  P V4  $\implies$   $\neg$  P V5  $\implies$   $\neg$  P V6  $\implies$   $\neg$  P V7
 $\implies$   $\neg$  P V8  $\implies$ 
       $\neg$  P V10  $\implies$  P V9
    proof(cases x rule: FV-cases, auto) qed
  qed
qed

end

notepad
begin
  let ?G = ( $\text{verts} = \{V1, V2, V3, V4, V5, V6, V7, V8\}$ ,  $\text{arcs} = \{(V2, V1), (V3, V2), (V4, V2), (V5, V2),$ 
     $(V6, V1), (V7, V6), (V8, V7)\}$ ,  $\text{tail} = \text{fst}$ ,  $\text{head} = \text{snd}$ )
  value blockDAG ?G
  value OrderDAG ?G 2
  let ?G2 = ( $\text{verts} = \{V1, V2, V3, V4, V5, V6, V7, V8, V9\}$ ,  $\text{arcs} = \{(V2, V1), (V3, V2), (V4, V2), (V5, V2),$ 
     $(V6, V1), (V7, V6), (V8, V7), (V9, V3), (V9, V4), (V9, V5), (V9, V8)\}$ ,  $\text{tail} = \text{fst}$ ,  $\text{head}$ 
    =  $\text{snd}$ )
  value blockDAG ?G2
  value OrderDAG ?G2 2
  value Append-One ?G ?G2 V9
  value Honest-Append-One ?G ?G2 V9
  let ?G3 = ( $\text{verts} = \{V1, V2, V3, V4, V5, V6, V7, V8, V9, V10\}$ ,  $\text{arcs} = \{(V2, V1), (V3, V2), (V4, V2), (V5, V2),$ 
     $(V6, V1), (V7, V6), (V8, V7), (V9, V3), (V9, V4), (V9, V5), (V9, V8), (V10, V3), (V10, V4), (V10, V5)\}$ ,
     $\text{tail} = \text{fst}$ ,  $\text{head} = \text{snd}$ )
  value Append-One ?G2 ?G3 V10
  value Append-One-Honest-Dishonest ?G ?G2 V9 ?G3 V10
  value OrderDAG ?G3 2
  value (V6, V2)  $\in$  GHOSTDAG 2 ?G
  value (V6, V2)  $\notin$  GHOSTDAG 2 ?G3
end

end

```