# blockDAGs

Jörn

June 28, 2021

## Contents



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

# 1 Definitions

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

# 2 Functions

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

**fun** (**in** *DAG*) *future-nodes*:: $'a \Rightarrow 'a\ set$
  **where** *future-nodes* $a = \{b.\ \ b \rightarrow^{+}{}_{G}\ a\}$

**fun** (**in** *DAG*) *past-nodes*:: $'a \Rightarrow 'a\ set$
  **where** *past-nodes* $a = \{b.\ a \rightarrow^{+}{}_{G}\ b\}$

**fun** (**in** *DAG*) *past-nodes-refl* :: $'a \Rightarrow 'a\ set$
  **where** *past-nodes-refl* $a = \{b.\ a \rightarrow^{*}{}_{G}\ b\}$

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

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

**fun** (**in** *DAG*) *is-tip*:: $'a \Rightarrow bool$
  **where** *is-tip* $a = ((a \in verts\ G) \wedge\ (ALL\ x.\ \neg\ x \rightarrow^{+}\ a))$

**definition** (**in** *DAG*) *tips*:: $'a\ set$
  **where** *tips* = $\{v.\ is\text{-}tip\ v\}$

# 3 Lemmas

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

## 3.1 Tips

**lemma** (**in** *DAG*) *del-tips-dag*:
**assumes** *is-tip 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** $v \rightarrow^+_{del\text{-}vert\ t}\ v$
**then have** $v \rightarrow^+ v$ **using** *subgraph-del-vert*
**by** (*meson arcs-ends-mono trancl-mono*)
**then show** *False*
**by** (*simp add*: *cycle-free*)
**qed**

## 3.2 Future Nodes

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

## 3.3 Past Nodes

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

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

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

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

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

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

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

**lemma** (**in** *DAG*) *past-future-dis*[*simp*]: $past\text{-}nodes\ a \cap future\text{-}nodes\ a = \{\}$
**proof** (*rule ccontr*)

**assume** $\neg$ *past-nodes a* $\cap$ *future-nodes a* $= \{\}$
**then show** *False*
**using** *past-nodes.simps future-nodes.simps unidirectional reachable1-reachable*
**by** *blast*
**qed**

## 3.4 Reduce Past

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

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

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

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

## 3.5 Reduce Past Reflexiv

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

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

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

end

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

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

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

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

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

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

# 4 Definitions

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

# 5 Functions

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

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

# 6 Lemmas

**lemma** *subs*:
**assumes** *blockDAG G*

**shows** *DAG G* $\wedge$ *digraph G* $\wedge$ *fin-digraph G* $\wedge$ *wf-digraph G*
**using** *assms blockDAG-def DAG-def digraph-def fin-digraph-def* **by** *blast*

## 6.1 Genesis

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

**lemma** (**in** *blockDAG*) *genesis-existAlt*:
$\exists$ *a*. *is-genesis-node a*
**using** *genesis genesisAlt blockDAG-axioms-def* **by** *presburger*

**lemma** (**in** *blockDAG*) *unique-genesis*: *is-genesis-node a* $\wedge$ *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*. *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 someI2-ex*
**by** *metis*

## 6.2 Tips

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

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


lemma (in blockDAG) tips-unequal-gen:
  assumes card( verts G) > 1
  shows ∃ p. p ∈ verts G ∧ is-tip p ∧ ¬is-genesis-node p
proof –
  have b1: 1 < card (verts G) using assms 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 x by auto
  then obtain z where z-def: z ∈ (verts G) − {x} ∧ is-tip 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 show** *?thesis* **using** *z-def* **by** *auto*
**qed**

**lemma** (**in** *blockDAG*) *del-tips-bDAG*:
**assumes** *is-tip 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** $u \rightarrow^*_{pre\text{-}digraph.del\text{-}arc\ (del\text{-}vert\ t)\ e}\ v$
**then have** *path*: $u \rightarrow^*_{del\text{-}arc\ e}\ v$
**by** (*meson del-arc-subgraph subgraph-del-vert digraph-axioms*
*digraph-subgraph pre-digraph.reachable-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*: $g \in$ *verts* (*del-vert t*)
**using** *assms*(*2*) *genesis del-vert-simps* **by** *auto*
**have** ($\forall r.\ r \in$ *verts* (*del-vert t*) $\longrightarrow r \rightarrow^*_{del\text{-}vert\ t}\ g$)
**proof** *safe*
**fix** *r*
**assume** *in-del*: $r \in$ *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*: $t \notin$ (*set* ( *map* ($\lambda s.$ (*head G s*)) *p*))
**proof** (*rule ccontr*)
**assume** ¬ $t \notin$ (*set* ( *map* ($\lambda s.$ (*head G s*)) *p*))
**then have** *as*: $t \in$ (*set* ( *map* ($\lambda s.$ (*head G s*)) *p*))
**by** *auto*
**then obtain** *e* **where** *tl*: $t =$ (*head G e*) $\wedge\ e \in$ *arcs G*
**using** *wf-digraph-def awalk-def path* **by** *auto*
**then obtain** *u* **where** *hd*: $u =$ (*tail G e*) $\wedge\ u \in$ *verts G*
**using** *wf-digraph-def tl* **by** *auto*
**have** $t \in$ *verts G*
**using** *assms*(*1*) *is-tip.simps* **by** *blast*
**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) by auto
qed
have neither: r ≠ t ∧ g ≠ t
    using del-vert-def assms(2) gen in-del by auto
have no-tail: t ∉ (set ( map (tail G) p))
proof(rule ccontr)
  assume as2: ¬ t ∉ set (map (tail G) p)
  then have tl2: t ∈ set (map (tail G) p) by auto
  then have t ∈ set (map (head G) p)
  proof (induct rule: cas.induct)
    case (1 u v)
    then have v ∉ set (map (tail G) []) by auto
    then show v ∈ set (map (tail G) []) ⟹ v ∈ set (map (head G) [])
      by auto
  next
    case (2 u e es v)
    then show ?case
      using set-awalk-verts-not-Nil-cas neither awalk-def cas.simps(2) path
      by (metis UnCI tl2 awalk-verts-conv'
          cas-simp list.simps(8) no-head set-ConsD)
  qed
  then show False using no-head by auto
qed
have pre-digraph.awalk (del-vert t) r p g
  unfolding pre-digraph.awalk-def
proof safe
  show r ∈ verts (del-vert t) using in-del by simp
next
  fix x
  assume as3: x ∈ set p
  then have ht: head G x ≠ t ∧ tail G x ≠ t
    using no-head no-tail by auto
  have  x ∈ arcs G
    using awalk-def path subsetD as3 by auto
  then show x ∈ arcs (del-vert t) using del-vert-simps(2) ht by auto
next
  have pre-digraph.cas G r p g using path by auto
  then show pre-digraph.cas (del-vert t) r p g
  proof(induct p arbitrary:r)
    case Nil
    then have r = g using awalk-def cas.simps by auto
    then show ?case using pre-digraph.cas.simps(1)
      by (metis)
  next
    case (Cons a p)
    assume pre: ⋀r. (cas r p g ⟹ pre-digraph.cas (del-vert t) r p g)
    and one: cas r (a # p) g
```

**then have** *two*: *cas* (*head G a*) *p g*
**using** *awalk-def* **by** *auto*
**then have** *t*: *tail* (*del-vert t*) *a* = *r*
**using** *one cas.simps awalk-def del-vert-simps*(*3*) **by** *auto*
**then show** *?case*
**unfolding** *pre-digraph.cas.simps*(*2*) *t*
**using** *pre two del-vert-simps*(*4*) **by** *auto*
**qed**
**qed**
**then show** $r \rightarrow^*_{del\text{-}vert\ t}\ g$ **by** (*meson wf-digraph.reachable-awalkI del-tips-dag assms*(*1*) *DAG-def digraph-def fin-digraph-def*)
**qed**
**then show** $\exists p.\ p \in verts\ (del\text{-}vert\ t) \wedge (\forall r.\ r \in verts\ (del\text{-}vert\ t) \longrightarrow r \rightarrow^*_{del\text{-}vert\ t}\ p)$
**using** *gen genp* **by** *auto*
**qed**

# 7 Future Nodes

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

## 7.1 Reduce Past

**lemma** (**in** *blockDAG*) *reduce-past-not-empty*:
**assumes** $a \in verts\ G$
**and** $\neg is\text{-}genesis\text{-}node\ a$
**shows** $(verts\ (reduce\text{-}past\ a)) \neq \{\}$
**proof** –
**obtain** *g*
**where** *gen*: *is-genesis-node g* **using** *genesis-existAlt* **by** *auto*
**have** *ex*: $g \in verts\ (reduce\text{-}past\ a)$ **using** *reduce-past.simps past-nodes.simps genesisAlt reachable-neq-reachable1 reachable-reachable1-trans gen assms*(*1*) *assms*(*2*) **by** *auto*
**then show** $(verts\ (reduce\text{-}past\ a)) \neq \{\}$ **using** *ex* **by** *auto*
**qed**

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

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

**proof** *safe*
  **show** *digraph* (*reduce-past a*)
    **using** *digraphI-induced reduce-past-induced-subgraph* **by** *auto*
**next**
  **show** *DAG-axioms* (*reduce-past a*)
    **unfolding** *DAG-axioms-def*
    **using** *cycle-free reduce-past-path* **by** *metis*
**next**
  **show** *blockDAG-axioms* (*reduce-past a*)
  **unfolding** *blockDAG-axioms-def*
  **proof** *safe*
    **fix** *u v e*
    **assume** *arc*: *wf-digraph.arc* (*reduce-past a*) *e* (*u*, *v*)
    **then show** $u \rightarrow^*{}_{\textit{pre-digraph.del-arc (reduce-past a) e}}\ v \Longrightarrow \textit{False}$
    **proof** −
        **assume** *e-in*: (*wf-digraph.arc* (*reduce-past 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 a*)
        **using** *reduce-past.simps subgraph-def subgraph-refl wf-digraph.wellformed-induce-subgraph*
            **by** *metis*
          **then have** $e \in$ *arcs* (*reduce-past a*) ∧ *tail* (*reduce-past a*) $e = u$
                    ∧ *head* (*reduce-past a*) $e = v$
            **using** *arc wf-digraph.arcE*
            **by** *metis*
          **then show** *?thesis*
            **using** *arc-def reduce-past.simps* **by** *auto*
        **qed**
        **then have** $\neg\ u \rightarrow^*{}_{\textit{del-arc e}}\ v$
          **using** *only-new* **by** *auto*
        **then show** $u \rightarrow^*{}_{\textit{pre-digraph.del-arc (reduce-past a) e}}\ v \Longrightarrow \textit{False}$
          **using** *DAG.past-nodes-verts reduce-past.simps blockDAG-axioms subs*
              *del-arc-subgraph digraph.digraph-subgraph digraph-axioms*
              *pre-digraph.reachable-mono subgraph-induce-subgraphI*
          **by** *metis*
      **qed**
    **next**
        **obtain** *p* **where** *gen*: *is-genesis-node p* **using** *genesis-existAlt* **by** *auto*

**have** *pe*: $p \in verts\ (reduce\text{-}past\ a) \wedge (\forall r.\ r \in verts\ (reduce\text{-}past\ a) \longrightarrow r \rightarrow^*{}_{reduce\text{-}past\ a}\ p)$
**proof**
**show** $p \in verts\ (reduce\text{-}past\ a)$ **using** *genesisAlt induce-reachable-preserves-paths reduce-past.simps past-nodes.simps reachable1-reachable induce-subgraph-verts assms(2) assms(3) gen mem-Collect-eq reachable-neq-reachable1*
**by** (*metis* (*no-types*, *lifting*))

**next**
**show** $\forall r.\ r \in verts\ (reduce\text{-}past\ a) \longrightarrow r \rightarrow^*{}_{reduce\text{-}past\ a}\ p$
**proof** *safe*
**fix** *r a*
**assume** *in-past*: $r \in verts\ (reduce\text{-}past\ a)$
**then have** *con*: $r \rightarrow^* p$ **using** *gen genesisAlt past-nodes-verts* **by** *auto*
**then show** $r \rightarrow^*{}_{reduce\text{-}past\ a}\ p$
**proof** −
**have** *f1*: $r \in verts\ G \wedge a \rightarrow^+ r$
**using** *in-past past-nodes-verts* **by** *force*
**obtain** *aaa* :: $'a\ set \Rightarrow\ 'a\ set \Rightarrow\ 'a$ **where**
*f2*: $\forall x0\ x1.\ (\exists v2.\ v2 \in x1 \wedge v2 \notin x0) = (aaa\ x0\ x1 \in x1 \wedge aaa\ x0\ x1 \notin x0)$
**by** *moura*
**have** $r \rightarrow^* aaa\ (past\text{-}nodes\ a)\ (Collect\ (reachable\ G\ r))$
$\longrightarrow a \rightarrow^+ aaa\ (past\text{-}nodes\ a)\ (Collect\ (reachable\ G\ r))$
**using** *f1* **by** (*meson reachable1-reachable-trans*)
**then have** $aaa\ (past\text{-}nodes\ a)\ (Collect\ (reachable\ G\ r)) \notin Collect\ (reachable\ G\ r)$
$\vee\ aaa\ (past\text{-}nodes\ a)\ (Collect\ (reachable\ G\ r)) \in past\text{-}nodes\ a$
**by** (*simp add*: *reachable-in-verts*(*2*))
**then have** $Collect\ (reachable\ G\ r) \subseteq past\text{-}nodes\ a$
**using** *f2* **by** (*meson subsetI*)
**then show** *?thesis*
**using** *con induce-reachable-preserves-paths reachable-induce-ss reduce-past.simps*
**by** (*metis* (*no-types*))
**qed**
**qed**
**qed**
**show**
$\exists p.\ p \in verts\ (reduce\text{-}past\ a) \wedge (\forall r.\ r \in verts\ (reduce\text{-}past\ a) \longrightarrow r \rightarrow^*{}_{reduce\text{-}past\ a}\ p)$
**using** *pe* **by** *auto*
**qed**
**qed**

## 7.2 Reduce Past Reflexiv

**lemma** (**in** *blockDAG*) *reduce-past-refl-induced-subgraph*:

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

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

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

**lemma** (**in** *blockDAG*) *reduce-past-refl-dagbased*:
**assumes** $a \in$ *verts G*
**shows** *blockDAG* (*reduce-past-refl a*)
**unfolding** *blockDAG-def DAG-def*
**proof** *safe*
**show** *digraph* (*reduce-past-refl a*)
**using** *reduce-past-refl-digraph assms*(*1*) **by** *simp*
**next**
**show** *DAG-axioms* (*reduce-past-refl 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 a*)
**unfolding** *blockDAG-axioms*
**proof**
**fix** *u v*
**show** $\forall$ *e*. $u \rightarrow^*{}_{\textit{pre-digraph.del-arc (reduce-past-refl a) e}}\ v \longrightarrow \neg$ *wf-digraph.arc* (*reduce-past-refl a*) *e* (*u*, *v*)
**proof** *safe*
**fix** *e*
**assume** *a*: *wf-digraph.arc* (*reduce-past-refl a*) *e* (*u*, *v*)
**and** *b*: $u \rightarrow^*{}_{\textit{pre-digraph.del-arc (reduce-past-refl a) e}}\ v$
**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 a*)
**using** *reduce-past-refl-digraph digraph-def* **by** *auto*
**then have** $e \in$ *arcs* (*reduce-past-refl a*) $\wedge$ *tail* (*reduce-past-refl a*) $e = u$ $\wedge$ *head* (*reduce-past-refl a*) $e = v$
**using** *wf-digraph.arcE arc-def a*
**by** (*metis* (*no-types*))
**then show** *arc e* (*u*, *v*)
**using** *arc-def reduce-past-refl.simps* **by** *auto*
**qed**
**have** $u \rightarrow^*{}_{\textit{pre-digraph.del-arc G e}}\ v$
**using** *a b reduce-past-refl-digraph del-arc-subgraph digraph-axioms*

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

## 7.3 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-digraph:
  digraph gen-graph
using digraphI-induced induced-induce gen-graph-def
       genesis-in-verts by simp

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


lemma (in blockDAG) gen-graph-sound:
  blockDAG (gen-graph)
  unfolding blockDAG-def DAG-def blockDAG-axioms-def
proof safe
   show digraph gen-graph using gen-graph-digraph by simp
 next
   have (arcs-ends gen-graph)⁺ = {}
     using trancl-empty gen-graph-empty-arcs by (simp add: arcs-ends-def)
  then show DAG-axioms gen-graph
    by (simp add: DAG-axioms.intro)
next
  fix u v e
  have wf-digraph.arc gen-graph e (u, v) ≡ 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* $\rightarrow^*{}_{gen\text{-}graph}$ *genesis-node*
    **using** *gen-gen rtrancl-on-refl*
    **by** (*simp add*: *reachable-def*)
  **have** $\forall r.\ r \in$ *verts gen-graph* $\longrightarrow r \rightarrow^*{}_{gen\text{-}graph}$ *genesis-node*
  **proof** *safe*
    **fix** *r*
    **assume** $r \in$ *verts gen-graph*
    **then have** $r =$ *genesis-node*
      **using** *gen-gen* **by** *auto*
    **then show** $r \rightarrow^*{}_{gen\text{-}graph}$ *genesis-node*
      **by** (*simp add*: *local.refl*)
  **qed**
  **then show** $\exists p.\ p \in$ *verts gen-graph* $\wedge$
      ($\forall r.\ r \in$ *verts gen-graph* $\longrightarrow r \rightarrow^*{}_{gen\text{-}graph}\ p$)
    **by** (*simp add*: *gen-gen*)
**qed**

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

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

  **show** *P Z*
    **using** *bG bD*
  **proof** (*induction card* (*verts Z*) *arbitrary*: *Z rule*: *Nat.nat-induct-non-zero*)
    **case** *1*
    **then show** *?case* **using** *cases*(*1*) **by** *auto*
**next**
  **case** *su*: (*Suc n*)
  **show** *?case*
    **by** (*metis local.cases*(*2*) *su.hyps*(*2*) *su.hyps*(*3*) *su.prems*)
  **qed**

**qed**

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

**qed**

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

**lemma** (**in** *blockDAG*) *blockDAG-induct*[*consumes 1*, *case-names base step*]:
**assumes** *cases*: $\bigwedge V$. *blockDAG V* $\Longrightarrow$ *P* (*blockDAG.gen-graph V*)
$\bigwedge H$.
($\bigwedge b$. *blockDAG* (*pre-digraph.del-vert H b*) $\Longrightarrow$ *b* $\in$ *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** *blockDAG-axioms* **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* $\Longrightarrow$ *card* (*verts V*) = *c* $\Longrightarrow$ *P V*) $\Longrightarrow$
*blockDAG W* $\Longrightarrow$ *card* (*verts W*) = *Suc c* $\Longrightarrow$ *P W*
**proof** −
**assume** *ind*: $\bigwedge V$. (*blockDAG V* $\Longrightarrow$ *card* (*verts V*) = *c* $\Longrightarrow$ *P V*)
**and** *bD*: *blockDAG W*
**and** *size*: *card* (*verts W*) = *Suc c*
**have** *assm2*: $\bigwedge b$. *blockDAG* (*pre-digraph.del-vert W b*)
$\Longrightarrow$ *b* $\in$ *verts W* $\Longrightarrow$ *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 fin-digraph-del-vert*
*size*
**by** (*simp add*: *fin-digraph.finite-verts*
*DAG.axioms blockDAG.axioms 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**

**end**

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

# 8 Definitions

**locale** *tie-breakingDAG* =
  **fixes** *G*::(*'a*::*linorder*,*'b*) *pre-digraph*
  **assumes** *is-blockDAG*: *blockDAG G*

# 9 Functions

**definition** *tie-break-int*:: *'a*::*linorder* ⇒ *'a* ⇒ *int* ⇒ *int*
  **where** *tie-break-int a b i* =
 (*if i=0 then* (*if* (*a* ≤ *b*) *then 1 else* −*1*) *else*
     (*if i* > *0 then 1 else* −*1*))

**fun** *sumlist-acc* :: *'a*::*linorder* ⇒*'a* ⇒ *int* ⇒ *int list* ⇒ *int*
  **where** *sumlist-acc a b s* [] = *tie-break-int a b s*
  | *sumlist-acc a b s* (*x*#*xs*) = *sumlist-acc a b* (*s* + *x*) *xs*

**fun** *sumlist* :: *'a*::*linorder* ⇒*'a* ⇒ *int list* ⇒ *int*
  **where** *sumlist a b* [] = *0*
  | *sumlist a b* (*x* # *xs*) = *sumlist-acc a b 0* (*x* # *xs*)

**function** *vote-Spectre* :: (*'a*::*linorder*,*'b*) *pre-digraph* ⇒*'a* ⇒ *'a* ⇒ *'a* ⇒ *int*
  **where**
  *vote-Spectre V a b c* = (
  *if* (¬ *blockDAG V* ∨ *a* ∉ *verts V* ∨ *b* ∉ *verts V* ∨ *c* ∉ *verts V*) *then 0 else*
  *if* (*b*=*c*) *then 1 else*
  *if* ((*a* $\rightarrow^*{}_V$ *b*) ∧ ¬(*a* $\rightarrow^+{}_V$ *c*)) *then 1 else*
  *if* ((*a* $\rightarrow^*{}_V$ *c*) ∧ ¬(*a* $\rightarrow^+{}_V$ *b*)) *then* −*1 else*
  *if* ((*a* $\rightarrow^+{}_V$ *b*) ∧ (*a* $\rightarrow^+{}_V$ *c*)) *then*
   (*sumlist b c* (*map* (λ*i*.
 (*vote-Spectre* (*DAG.reduce-past V a*) *i b c*)) (*sorted-list-of-set* ((*DAG.past-nodes V a*)))))
 *else*
   *sumlist b c* (*map* (λ*i*.
   (*vote-Spectre V i b c*)) (*sorted-list-of-set* (*DAG.future-nodes V a*))))
  **by** *auto*
**termination**
**proof**
**let** *?R* = *measures* [( λ(*V*, *a*, *b*, *c*). (*card* (*verts V*))), ( λ(*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: ¬ (¬ blockDAG V ∨ a ∉ verts V ∨ b ∉ verts V ∨ c ∉ verts V)
  then have a ∈ verts V  by simp
  then have card (verts (DAG.reduce-past V a)) < card (verts V)
    using bD blockDAG.reduce-less
    by metis
  then show ((DAG.reduce-past V a, x, b, c), V, a, b, c)
      ∈ measures
          [λ(V, a, b, c). card (verts V),
           λ(V, a, b, c). card {e. e →* V a}]
    by simp
next
  fix V::('a::linorder, 'b) pre-digraph
  fix x a b c
  assume bD: ¬ (¬ blockDAG V ∨ a ∉ verts V ∨ b ∉ verts V ∨ c ∉ verts V)
  then have a-in: a ∈ verts V using bD by simp
  assume x ∈ set (sorted-list-of-set (DAG.future-nodes V a))
  then have x ∈ DAG.future-nodes V a using DAG.finite-future
    set-sorted-list-of-set bD subs
    by metis
  then have rr: x →+ V a using DAG.future-nodes.simps bD subs mem-Collect-eq
    by metis
  then have a-not: ¬ a →* V x using bD DAG.unidirectional subs by metis
  have bD2: blockDAG V using bD by simp
  have ∀x. {e. e →* V x} ⊆ verts V using subs bD2  subsetI
      wf-digraph.reachable-in-verts(1) mem-Collect-eq
    by metis
  then have fin: ∀x. finite {e. e →* V x} using subs bD2 fin-digraph.finite-verts
      finite-subset
    by metis
  have x →* V a using rr wf-digraph.reachable1-reachable subs bD2 by metis
  then have {e. e →* V x} ⊆ {e. e →* V a} using rr
      wf-digraph.reachable-trans Collect-mono subs bD2 by metis
  then have {e. e →* V x} ⊂ {e. e →* V a} using a-not
  subs bD2 a-in mem-Collect-eq psubsetI wf-digraph.reachable-refl
    by metis
  then have card {e. e →* V x} < card {e. e →* V a} using fin
    by (simp add: psubset-card-mono)
  then show ((V, x, b, c), V, a, b, c)
      ∈ measures
          [λ(V, a, b, c). card (verts V), λ(V, a, b, c). card {e. e →* V a}]
    by simp
qed

end
```