Minimal CBC Casper Isabelle/HOL proofs

LayerX

August 4, 2019

Contents

theory *Strict-Order*

imports *Main*

begin

notation *Set.empty* ($\emptyset$)

definition *strict-partial-order r* $\equiv$ *trans r* $\wedge$ *irrefl r*

definition *strict-well-order-on A r* $\equiv$ *strict-linear-order-on A r* $\wedge$ *wf r*

lemma *strict-linear-order-is-strict-partial-order* :
  strict-linear-order-on A r $\Longrightarrow$ *strict-partial-order r*
  by (*simp add*: *strict-linear-order-on-def strict-partial-order-def*)

definition *upper-bound-on* :: *'a set* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
  where
    upper-bound-on A r x = ($\forall$ *y*. *y* $\in$ *A* $\longrightarrow$ (*y*, *x*) $\in$ *r* $\vee$ *x* = *y*)

definition *maximum-on* :: *'a set* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
  where

maximum-on A r x = (*x* ∈ *A* ∧ *upper-bound-on A r x*)

definition *minimal-on* :: *'a set* ⇒ *'a rel* ⇒ *'a* ⇒ *bool*
where
minimal-on A r x = (*x* ∈ *A* ∧ (∀ *y*. (*y*, *x*) ∈ *r* ⟶ *y* ∉ *A*))

definition *maximal-on* :: *'a set* ⇒ *'a rel* ⇒ *'a* ⇒ *bool*
where
maximal-on A r x = (*x* ∈ *A* ∧ (∀ *y*. (*x*, *y*) ∈ *r* ⟶ *y* ∉ *A*))

lemma *maximal-and-maximum-coincide-for-strict-linear-order* :
strict-linear-order-on A r ⟹ *maximal-on A r x* = *maximum-on A r x*
apply (*simp add*: *strict-linear-order-on-def irrefl-def total-on-def trans-def maximal-on-def maximum-on-def upper-bound-on-def*)
by *blast*

lemma *strict-partial-order-on-finite-non-empty-set-has-maximal* :
strict-partial-order r ⟶ *finite A* ⟶ *A* ≠ ∅ ⟶ (∃ *x*. *maximal-on A r x*)
proof −
have ⋀*n*. *strict-partial-order r* ⟹ (∀ *A*. *Suc n* = *card A* ⟶ *finite A* ⟶ *A* ≠ ∅ ⟶ (∃ *x*. *maximal-on A r x*))
proof −
assume *strict-partial-order r*
then have (∀ *a*. (*a*, *a*) ∉ *r*)
by (*simp add*: *strict-partial-order-def irrefl-def*)
fix *n*
show ∀ *A*. *Suc n* = *card A* ⟶ *finite A* ⟶ *A* ≠ ∅ ⟶ (∃ *x*. *maximal-on A r x*)
apply (*induction n*)
unfolding *maximal-on-def*
using ‹(∀ *a*. (*a*, *a*) ∉ *r*)›
apply (*metis card-eq-SucD empty-iff insert-iff*)
proof −
fix *n*
assume ∀ *A*. *Suc n* = *card A* ⟶ *finite A* ⟶ *A* ≠ ∅ ⟶ (∃ *x*. *x* ∈ *A* ∧ (∀ *y*. (*x*, *y*) ∈ *r* ⟶ *y* ∉ *A*))
have ∀ *B*. *Suc* (*Suc n*) = *card B* ⟶ *finite B* ⟶ *B* ≠ ∅ ⟶ (∃ *A'* *b*. *B* = *A'* ∪ {*b*} ∧ *card A'* = *Suc n* ∧ *b* ∉ *A'*)
by (*metis Un-commute add-diff-cancel-left' card-gt-0-iff card-insert-disjoint card-le-Suc-iff insert-is-Un not-le not-less-eq-eq plus-1-eq-Suc*)
then have ∀ *B*. *Suc* (*Suc n*) = *card B* ⟶ *finite B* ⟶ *B* ≠ ∅ ⟶ (∃ *A'* *b*. *B* = *A'* ∪ {*b*} ∧ *card A'* = *Suc n* ∧ *finite A'* ∧ *A'* ≠ ∅ ∧ *b* ∉ *A'*)
by (*metis card-gt-0-iff zero-less-Suc*)
then have ∀ *B*. *Suc* (*Suc n*) = *card B* ⟶ *finite B* ⟶ *B* ≠ ∅
⟶ (∃ *A'* *b* *x*. *B* = *A'* ∪ {*b*} ∧ *b* ∉ *A'* ∧ *x* ∈ *A'* ∧ (∀ *y*. (*x*, *y*) ∈ *r* ⟶ *y* ∉ *A'*))
using ‹∀ *A*. *Suc n* = *card A* ⟶ *finite A* ⟶ *A* ≠ ∅ ⟶ (∃ *x*. *x* ∈ *A* ∧ (∀ *y*. (*x*, *y*) ∈ *r* ⟶ *y* ∉ *A*))›
by *metis*

```
      then show ∀B. Suc (Suc n) = card B ⟶ finite B ⟶ B ≠ ∅ ⟶ (∃x. x
∈ B ∧ (∀y. (x, y) ∈ r ⟶ y ∉ B))
      by (metis (no-types, lifting) Un-insert-right ‹∀a. (a, a) ∉ r› ‹strict-partial-order
r› insertE insert-iff strict-partial-order-def sup-bot.right-neutral transE)
    qed
  qed
  then show ?thesis
    by (metis card.insert-remove finite.cases)
qed

lemma strict-partial-order-has-at-most-one-maximum :
  strict-partial-order r
  ⟶ {x. maximum-on A r x} ≠ ∅
  ⟶ is-singleton {x. maximum-on A r x}
proof (rule ccontr)
  assume ¬ (strict-partial-order r ⟶ {x. maximum-on A r x} ≠ ∅ ⟶ is-singleton
{x. maximum-on A r x})
  then have strict-partial-order r ⟶ {x. maximum-on A r x} ≠ ∅ ⟶ ¬ is-singleton
{x. maximum-on A r x}
    by simp
  then have strict-partial-order r ⟶ {x. maximum-on A r x} ≠ ∅ ⟶ (∃ x1 x2.
x1 ≠ x2 ∧ {x1, x2} ⊆ {x. maximum-on A r x})
    by (meson empty-subsetI insert-subset is-singletonI')
  then have strict-partial-order r ⟶ {x. maximum-on A r x} ≠ ∅ ⟶ (∃ x1 x2.
x1 ≠ x2 ∧ {x1, x2} ⊆ {x ∈ A. ∀ y. y ∈ A ⟶ (y, x) ∈ r ∨ x = y})
    by (simp add: maximum-on-def upper-bound-on-def)
  then have strict-partial-order r ⟶ {x. maximum-on A r x} ≠ ∅ ⟶ (∃ x1 x2.
x1 ≠ x2 ∧ {x1, x2} ⊆ A ∧ (∀ y. y ∈ A ⟶ (y, x1) ∈ r ∨ x1 = y) ∧ (∀ y. y ∈
A ⟶ (y, x2) ∈ r ∨ x2 = y))
    by auto
  then show False
    using strict-partial-order-def

     by (metis ‹¬ (strict-partial-order r ⟶ {x. maximum-on A r x} ≠ ∅ ⟶
is-singleton {x. maximum-on A r x})› insert-subset irrefl-def transE)
qed

lemma strict-linear-order-on-finite-non-empty-set-has-one-maximum :
  strict-linear-order-on A r ⟶ finite A ⟶ A ≠ ∅ ⟶ is-singleton {x. maximum-on
A r x}
  using strict-linear-order-is-strict-partial-order strict-partial-order-on-finite-non-empty-set-has-maximal

      strict-partial-order-has-at-most-one-maximum maximal-and-maximum-coincide-for-strict-linear-order
  by fastforce
```

definition *upper-bound-on-non-strict* :: *'a set* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
  where
    upper-bound-on-non-strict A r x = ($\forall$ *y*. *y* $\in$ *A* $\longrightarrow$ (*y*, *x*) $\in$ *r*)

definition *maximum-on-non-strict* :: *'a set* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
  where
    maximum-on-non-strict A r x = (*x* $\in$ *A* $\wedge$ *upper-bound-on-non-strict A r x*)

definition *maximal-on-non-strict* :: *'a set* $\Rightarrow$ *'a rel* $\Rightarrow$ *'a* $\Rightarrow$ *bool*
  where
    maximal-on-non-strict A r x = (*x* $\in$ *A* $\wedge$ ($\forall$ *y*. *y* $\in$ *A* $\longrightarrow$ (*y*, *x*) $\in$ *r* $\vee$ (*x*, *y*)
$\notin$ *r*))

lemma *preorder-on-finite-non-empty-set-has-maximal* :
  preorder-on A r $\longrightarrow$ *finite A* $\longrightarrow$ *A* $\neq \emptyset$ $\longrightarrow$ ($\exists$ *x*. *maximal-on-non-strict A r x*)
proof –
  have $\bigwedge$*n*. *preorder-on A r* $\Longrightarrow$ ($\forall$ *A*. *Suc n* = *card A* $\longrightarrow$ *finite A* $\longrightarrow$ *A* $\neq \emptyset$
$\longrightarrow$ ($\exists$ *x*. *maximal-on-non-strict A r x*))

  proof –
    fix *n*
    assume *preorder-on A r*
   show $\forall$ *A*. *Suc n* = *card A* $\longrightarrow$ *finite A* $\longrightarrow$ *A* $\neq \emptyset$ $\longrightarrow$ ($\exists$ *x*. *maximal-on-non-strict*
A r x)
      apply (*induction n*)
      unfolding *maximal-on-non-strict-def*
       apply (*metis card-eq-SucD singletonD singletonI*)

    proof –
      fix *n*
     assume $\forall$*A*. *Suc n* = *card A* $\longrightarrow$ *finite A* $\longrightarrow$ *A* $\neq \emptyset$ $\longrightarrow$ ($\exists$*x*. *x* $\in$ *A* $\wedge$ ($\forall$*y*.
y $\in$ *A* $\longrightarrow$ (*y*, *x*) $\in$ *r* $\vee$ (*x*, *y*) $\notin$ *r*))
      have $\forall$*B*. *Suc* (*Suc n*) = *card B* $\longrightarrow$ *finite B* $\longrightarrow$ *B* $\neq \emptyset$ $\longrightarrow$ ($\exists$ *A' b*. *B* =
A' $\cup$ {*b*} $\wedge$ *card A'* = *Suc n* $\wedge$ *b* $\notin$ *A'*)
        by (*metis Un-commute add-diff-cancel-left' card-gt-0-iff card-insert-disjoint*
card-le-Suc-iff insert-is-Un not-le not-less-eq-eq plus-1-eq-Suc)
      then have $\forall$*B*. *Suc* (*Suc n*) = *card B* $\longrightarrow$ *finite B* $\longrightarrow$ *B* $\neq \emptyset$
          $\longrightarrow$ ($\exists$ *A' b*. *B* = *A'* $\cup$ {*b*} $\wedge$ *card A'* = *Suc n* $\wedge$ *finite A'* $\wedge$ *A'* $\neq \emptyset$ $\wedge$ *b*
$\notin$ *A'*)
        by (*metis card-gt-0-iff zero-less-Suc*)
      then have $\forall$*B*. *Suc* (*Suc n*) = *card B* $\longrightarrow$ *finite B* $\longrightarrow$ *B* $\neq \emptyset$
          $\longrightarrow$ ($\exists$ *A' b x*. *B* = *A'* $\cup$ {*b*} $\wedge$ *b* $\notin$ *A'* $\wedge$ *x* $\in$ *A'* $\wedge$ ($\forall$*y*. *y* $\in$ *A'* $\longrightarrow$ (*y*,
x) $\in$ *r* $\vee$ (*x*, *y*) $\notin$ *r*))
        using ‹$\forall$*A*. *Suc n* = *card A* $\longrightarrow$ *finite A* $\longrightarrow$ *A* $\neq \emptyset$ $\longrightarrow$ ($\exists$*x*. *x* $\in$ *A* $\wedge$ ($\forall$*y*.
y $\in$ *A* $\longrightarrow$ (*y*, *x*) $\in$ *r* $\vee$ (*x*, *y*) $\notin$ *r*))›
        by *metis*
      then show $\forall$*B*. *Suc* (*Suc n*) = *card B* $\longrightarrow$ *finite B* $\longrightarrow$ *B* $\neq \emptyset$ $\longrightarrow$ ($\exists$*x*. *x*
$\in$ *B* $\wedge$ ($\forall$*y*. *y* $\in$ *B* $\longrightarrow$ (*y*, *x*) $\in$ *r* $\vee$ (*x*, *y*) $\notin$ *r*))

```
        by (metis (no-types, lifting) Un-insert-right ‹preorder-on A r› insertE
insert-iff preorder-on-def sup-bot.right-neutral transE)
    qed
  qed
  then show ?thesis
    by (metis card.insert-remove finite.cases)
qed
```

lemma *connex-preorder-on-finite-non-empty-set-has-maximum* :
 preorder-on A r $\wedge$ *total-on A r* $\longrightarrow$ *finite A* $\longrightarrow$ *A* $\neq \emptyset$ $\longrightarrow$ ($\exists$ *x. maximum-on-non-strict A r x*)
 apply (*simp add*: *total-on-def maximum-on-non-strict-def upper-bound-on-non-strict-def maximal-on-non-strict-def*)
 by (*metis maximal-on-non-strict-def order-on-defs*(*1*) *preorder-on-finite-non-empty-set-has-maximal refl-onD*)

end

1 CBC Casper

theory *CBCCasper*

imports *Main HOL.Real Libraries/Strict-Order Libraries/Restricted-Predicates Libraries/LaTeXsugar*

begin

notation *Set.empty* ($\emptyset$)

typedecl *validator*

typedecl *consensus-value*

datatype *message* =
 Message consensus-value $*$ *validator* $*$ *message list*

type-synonym *state* = *message set*

fun *sender* :: *message* ⇒ *validator*
  where
    sender (*Message* (-, *v*, -)) = *v*

fun *est* :: *message* ⇒ *consensus-value*
  where
    est (*Message* (*c*, -, -)) = *c*

fun *justification* :: *message* ⇒ *state*
  where
    justification (*Message* (-, -, *s*)) = *set s*

fun
  Σi :: (*validator set* × *consensus-value set* × (*message set* ⇒ *consensus-value set*)) ⇒ *nat* ⇒ *state set* **and**
  Mi :: (*validator set* × *consensus-value set* × (*message set* ⇒ *consensus-value set*)) ⇒ *nat* ⇒ *message set*
  where
    $\Sigma i\ (V,C,\varepsilon)\ 0 = \{\emptyset\}$
  | $\Sigma i\ (V,C,\varepsilon)\ n = \{\sigma \in Pow\ (Mi\ (V,C,\varepsilon)\ (n - 1)).\ finite\ \sigma \wedge (\forall\ m.\ m \in \sigma \longrightarrow justification\ m \subseteq \sigma)\}$
  | $Mi\ (V,C,\varepsilon)\ n = \{m.\ est\ m \in C \wedge sender\ m \in V \wedge justification\ m \in (\Sigma i\ (V,C,\varepsilon)\ n) \wedge est\ m \in \varepsilon\ (justification\ m)\}$

locale *Params* =
  fixes *V* :: *validator set*
  and *W* :: *validator* ⇒ *real*
  and *t* :: *real*
  fixes *C* :: *consensus-value set*
  and ε :: *message set* ⇒ *consensus-value set*

begin
  definition $\Sigma = (\bigcup i{\in}\mathbb{N}.\ \Sigma i\ (V,C,\varepsilon)\ i)$
  definition $M = (\bigcup i{\in}\mathbb{N}.\ Mi\ (V,C,\varepsilon)\ i)$
  definition *is-valid-estimator* :: (*state* ⇒ *consensus-value set*) ⇒ *bool*
    where
      $is\text{-}valid\text{-}estimator\ e = (\forall \sigma \in \Sigma.\ e\ \sigma \in Pow\ C - \{\emptyset\})$

  lemma Σi*-subset-Mi*: $\Sigma i\ (V,C,\varepsilon)\ (n + 1) \subseteq Pow\ (Mi\ (V,C,\varepsilon)\ n)$
    by *force*

  lemma Σi*-subset-to-Mi*: $\Sigma i\ (V,C,\varepsilon)\ n \subseteq \Sigma i\ (V,C,\varepsilon)\ (n{+}1) \Longrightarrow Mi\ (V,C,\varepsilon)\ n \subseteq Mi\ (V,C,\varepsilon)\ (n{+}1)$
    by *auto*

  lemma *Mi-subset-to-*Σi: $Mi\ (V,C,\varepsilon)\ n \subseteq Mi\ (V,C,\varepsilon)\ (n{+}1) \Longrightarrow \Sigma i\ (V,C,\varepsilon)$

```
(n+1) ⊆ Σi (V,C,ε) (n+2)
    by auto

  lemma Σi-monotonic: Σi (V,C,ε) n ⊆ Σi (V,C,ε) (n+1)
    apply (induction n)
    apply simp
   apply (metis Mi-subset-to-Σi Suc-eq-plus1 Σi-subset-to-Mi add.commute add-2-eq-Suc)
    done

  lemma Mi-monotonic: Mi (V,C,ε) n ⊆ Mi (V,C,ε) (n+1)
    apply (induction n)
    defer
    using Σi-monotonic Σi-subset-to-Mi apply blast
    apply auto
    done

  lemma Σi-monotonicity: ∀ m ∈ ℕ. ∀ n ∈ ℕ. m ≤ n ⟶ Σi (V,C,ε) m ⊆ Σi
(V,C,ε) n
    using Σi-monotonic
    by (metis Suc-eq-plus1 lift-Suc-mono-le)

  lemma Mi-monotonicity: ∀ m ∈ ℕ. ∀ n ∈ ℕ. m ≤ n ⟶ Mi (V,C,ε) m ⊆ Mi
(V,C,ε) n
    using Mi-monotonic
    by (metis Suc-eq-plus1 lift-Suc-mono-le)

  lemma message-is-in-Mi :
   ∀ m ∈ M. ∃ n ∈ ℕ. m ∈ Mi (V, C, ε) (n − 1)
    apply (simp add: M-def Σi.elims)
    by (metis Nats-1 Nats-add One-nat-def diff-Suc-1 plus-1-eq-Suc)

  lemma state-is-in-pow-Mi :
   ∀ σ ∈ Σ. (∃ n ∈ ℕ. σ ∈ Pow (Mi (V, C, ε) (n − 1)) ∧ (∀ m ∈ σ. justification
m ⊆ σ))
    apply (simp add: Σ-def)


    apply auto
    proof −
      fix y :: nat and σ :: message set
      assume a1: σ ∈ Σi (V, C, ε) y
      assume a2: y ∈ ℕ
      have σ ⊆ Mi (V, C, ε) y
          using a1 by (meson Params.Σi-monotonic Params.Σi-subset-Mi Pow-iff
contra-subsetD)
      then have ∃n. n ∈ ℕ ∧ σ ⊆ Mi (V, C, ε) (n − 1)
        using a2 by (metis (no-types) Nats-1 Nats-add diff-Suc-1 plus-1-eq-Suc)
      then show ∃n∈ℕ. σ ⊆ {m. est m ∈ C ∧ sender m ∈ V ∧ justification m
∈ Σi (V, C, ε) (n − Suc 0) ∧ est m ∈ ε (justification m)}
```

by *auto*
next
show $\bigwedge y\ \sigma\ m\ x.\ y \in \mathbb{N} \Longrightarrow \sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ y \Longrightarrow m \in \sigma \Longrightarrow x \in$ *justification* $m \Longrightarrow x \in \sigma$
using *Params.Σi-monotonic* **by** *fastforce*
qed

lemma *message-is-in-Mi-n* :
$\forall\ m \in M.\ \exists\ n \in \mathbb{N}.\ m \in Mi\ (V,\ C,\ \varepsilon)\ n$
by (*smt Mi-monotonic Suc-diff-Suc add-leE diff-add diff-le-self message-is-in-Mi neq0-conv plus-1-eq-Suc subsetCE zero-less-diff*)

lemma *message-in-state-is-valid* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \wedge m \in \sigma \longrightarrow\ m \in M$
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ m$
assume $\sigma \in \Sigma \wedge m \in \sigma$
have
$\exists\ n \in \mathbb{N}.\ m \in Mi\ (V,\ C,\ \varepsilon)\ n$
$\Longrightarrow m \in M$
using *M-def* **by** *blast*
then show
$m \in M$
apply (*simp add*: *M-def*)
by (*smt Mi.simps Params.Σi-monotonic PowD Suc-diff-Suc* ‹$\sigma \in \Sigma \wedge m \in \sigma$› *add-leE diff-add diff-le-self gr0I mem-Collect-eq plus-1-eq-Suc state-is-in-pow-Mi subsetCE zero-less-diff*)
qed

lemma *state-is-subset-of-M* : $\forall\ \sigma \in \Sigma.\ \sigma \subseteq M$
using *message-in-state-is-valid* **by** *blast*

lemma *state-is-finite* : $\forall\ \sigma \in \Sigma.$ *finite* σ
apply (*simp add*: *Σ-def*)
using *Params.Σi-monotonic* **by** *fastforce*

lemma *justification-is-finite* : $\forall\ m \in M.$ *finite* (*justification* m)
apply (*simp add*: *M-def*)
using *Params.Σi-monotonic* **by** *fastforce*

lemma *Σis-subseteq-of-pow-M*: $\Sigma \subseteq$ *Pow* M
by (*simp add*: *state-is-subset-of-M subsetI*)

lemma *M-type*: $\bigwedge m.\ m \in M \Longrightarrow$ *est* $m \in C \wedge$ *sender* $m \in V \wedge$ *justification* $m \in \Sigma$
unfolding *M-def Σ-def*
by *auto*

```
end


locale Protocol = Params +
  assumes V-type: V ≠ ∅ ∧ finite V
  and W-type: ∀ v ∈ V. W v > 0
  and t-type: 0 ≤ t t < sum W V
  and C-type: card C > 1
  and ε-type: is-valid-estimator ε

lemma (in Protocol) estimates-are-non-empty: ⋀ σ. σ ∈ Σ ⟹ ε σ ≠ ∅
  using is-valid-estimator-def ε-type by auto

lemma (in Protocol) estimates-are-subset-of-C: ⋀ σ. σ ∈ Σ ⟹ ε σ ⊆ C
  using is-valid-estimator-def ε-type by auto

lemma (in Params) empty-set-exists-in-Σ-0: ∅ ∈ Σi (V, C, ε) 0
  by simp

lemma (in Params) empty-set-exists-in-Σ: ∅ ∈ Σ
  apply (simp add: Σ-def)
  using Nats-0 Σi.simps(1) by blast

lemma (in Params) Σi-is-non-empty: Σi (V, C, ε) n ≠ ∅
  apply (induction n)
  using empty-set-exists-in-Σ-0 by auto

lemma (in Params) Σis-non-empty: Σ ≠ ∅
  using empty-set-exists-in-Σ by blast

lemma (in Protocol) estimates-exists-for-empty-set :
  ε ∅ ≠ ∅
  by (simp add: empty-set-exists-in-Σ estimates-are-non-empty)

lemma (in Protocol) non-justifying-message-exists-in-M-0:
  ∃ m. m ∈ Mi (V, C, ε) 0 ∧ justification m = ∅
  apply auto
proof -
  have ε ∅ ⊆ C
    using Params.empty-set-exists-in-Σ ε-type is-valid-estimator-def by auto
  then show ∃m. est m ∈ C ∧ sender m ∈ V ∧ justification m = ∅ ∧ est m ∈ ε
(justification m) ∧ justification m = ∅
    by (metis V-type all-not-in-conv est.simps estimates-exists-for-empty-set justi-
fication.simps sender.simps set-empty subsetCE)
qed

lemma (in Protocol) Mi-is-non-empty: Mi (V, C, ε) n ≠ ∅
  apply (induction n)
  using non-justifying-message-exists-in-M-0 apply auto
```

using *Mi-monotonic empty-iff empty-subsetI* **by** *fastforce*

lemma (**in** *Protocol*) *Mis-non-empty*: $M \neq \emptyset$
using *non-justifying-message-exists-in-M-0 M-def Nats-0* **by** *blast*

lemma (**in** *Protocol*) *C-is-not-empty* : $C \neq \emptyset$
using *C-type* **by** *auto*

lemma (**in** *Params*) Σ*i-is-subset-of-*Σ :
$\forall\ n \in \mathbb{N}.\ \Sigma i\ (V,\ C,\ \varepsilon)\ n \subseteq \Sigma$
by (*simp add*: Σ*-def SUP-upper*)

lemma (**in** *Protocol*) *message-justifying-state-in-*Σ*-n-exists-in-M-n* :
$\forall\ n \in \mathbb{N}.\ (\forall\ \sigma.\ \sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n \longrightarrow (\exists\ m.\ m \in Mi\ (V,\ C,\ \varepsilon)\ n \wedge$ *justification* $m = \sigma))$
apply *auto*
proof –
fix $n\ \sigma$
assume $n \in \mathbb{N}$
and $\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$
then have $\sigma \in \Sigma$
using Σ*i-is-subset-of-*Σ **by** *auto*
have $\varepsilon\ \sigma \neq \emptyset$
using *estimates-are-non-empty* ‹$\sigma \in \Sigma$› **by** *auto*
have *finite* σ
using *state-is-finite* ‹$\sigma \in \Sigma$› **by** *auto*
moreover have $\exists\ m.$ *sender* $m \in V \wedge$ *est* $m \in \varepsilon\ \sigma \wedge$ *justification* $m = \sigma$
using *est.simps sender.simps justification.simps V-type* ‹$\varepsilon\ \sigma \neq \emptyset$› ‹*finite* σ›
by (*metis all-not-in-conv finite-list*)
moreover have $\varepsilon\ \sigma \subseteq C$
using *estimates-are-subset-of-C* Σ*i-is-subset-of-*Σ ‹$n \in \mathbb{N}$› ‹$\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$› **by** *blast*
ultimately show $\exists\ m.$ *est* $m \in C \wedge$ *sender* $m \in V \wedge$ *justification* $m \in \Sigma i\ (V,\ C,\ \varepsilon)\ n \wedge$ *est* $m \in \varepsilon$ (*justification* m) $\wedge$ *justification* $m = \sigma$
using *Nats-1 One-nat-def*
using ‹$\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$› **by** *blast*
qed

lemma (**in** *Protocol*) Σ*-type*: $\Sigma \subset$ *Pow* M
proof –
obtain m **where** $m \in Mi\ (V,\ C,\ \varepsilon)\ 0 \wedge$ *justification* $m = \emptyset$
using *non-justifying-message-exists-in-M-0* **by** *auto*
then have $\{m\} \in \Sigma i\ (V,\ C,\ \varepsilon)$ (*Suc 0*)
using *Params.*Σ*i-subset-Mi* **by** *auto*
then have $\exists\ m'.\ m' \in\ Mi\ (V,\ C,\ \varepsilon)$ (*Suc 0*) $\wedge$ *justification* $m' = \{m\}$
using *message-justifying-state-in-*Σ*-n-exists-in-M-n Nats-1 One-nat-def* **by** *metis*
then obtain m' **where** $m' \in\ Mi\ (V,\ C,\ \varepsilon)$ (*Suc 0*) $\wedge$ *justification* $m' = \{m\}$ **by** *auto*

```
  then have {m'} ∈ Pow M
    using M-def
    by (metis Nats-1 One-nat-def PowD PowI Pow-bottom UN-I insert-subset)
  moreover have {m'} ∉ Σ
    using Params.state-is-in-pow-Mi Protocol-axioms ‹m' ∈ Mi (V, C, ε) (Suc 0)
∧ justification m' = {m}› by fastforce
  ultimately show ?thesis
    using Σis-subseteq-of-pow-M by auto
qed


lemma (in Protocol) M-type-counterexample:
  (∀ σ. ε σ = C) ⟹ M = {m. est m ∈ C ∧ sender m ∈ V ∧ justification m ∈
Σ}
  apply (simp add: M-def)
  apply auto
  using Σi-is-subset-of-Σ apply blast
  by (simp add: Σ-def)


definition observed :: message set ⇒ validator set
  where
    observed σ = {sender m | m. m ∈ σ}

lemma (in Protocol) observed-type :
  ∀ σ ∈ Pow M. observed σ ∈ Pow V
  using Params.M-type Protocol-axioms observed-def by fastforce

lemma (in Protocol) observed-type-for-state :
  ∀ σ ∈ Σ. observed σ ⊆ V
 using Params.M-type Protocol-axioms observed-def state-is-subset-of-M by fastforce


fun is-future-state :: (state ∗ state) ⇒ bool
  where
    is-future-state (σ1, σ2) = (σ1 ⊆ σ2)

lemma (in Params) state-difference-is-valid-message :
  ∀ σ σ'. σ ∈ Σ ∧ σ' ∈ Σ
  ⟶ is-future-state(σ, σ')
  ⟶ σ' − σ ⊆ M
  using state-is-subset-of-M by blast


definition justified :: message ⇒ message ⇒ bool
  where
    justified m1 m2 = (m1 ∈ justification m2)
```

definition *equivocation* :: (*message* ∗ *message*) ⇒ *bool*
  where
    equivocation =
      $(\lambda(m1,\ m2).\ sender\ m1\ =\ sender\ m2\ \wedge\ m1\ \neq\ m2\ \wedge\ \neg\ (justified\ m1\ m2)\ \wedge$
$\neg\ (justified\ m2\ m1))$

definition *is-equivocating* :: *state* ⇒ *validator* ⇒ *bool*
  where
    $is\text{-}equivocating\ \sigma\ v\ =\ (\exists\ m1\ \in\ \sigma.\ \exists\ m2\ \in\ \sigma.\ equivocation\ (m1,\ m2)\ \wedge\ sender$
$m1\ =\ v)$

definition *equivocating-validators* :: *state* ⇒ *validator set*
  where
    $equivocating\text{-}validators\ \sigma\ =\ \{v\ \in\ observed\ \sigma.\ is\text{-}equivocating\ \sigma\ v\}$

lemma (**in** *Protocol*) *equivocating-validators-type* :
  $\forall\ \sigma\ \in\ \Sigma.\ equivocating\text{-}validators\ \sigma\ \subseteq\ V$
  using *observed-type-for-state equivocating-validators-def* **by** *blast*

lemma (**in** *Protocol*) *equivocating-validators-is-finite* :
  $\forall\ \sigma\ \in\ \Sigma.\ finite\ (equivocating\text{-}validators\ \sigma)$
  using *V-type equivocating-validators-type rev-finite-subset* **by** *blast*

definition (**in** *Params*) *equivocating-validators-paper* :: *state* ⇒ *validator set*
  where
    $equivocating\text{-}validators\text{-}paper\ \sigma\ =\ \{v\ \in\ V.\ is\text{-}equivocating\ \sigma\ v\}$

lemma (**in** *Protocol*) *equivocating-validators-is-equivalent-to-paper* :
  $\forall\ \sigma\ \in\ \Sigma.\ equivocating\text{-}validators\ \sigma\ =\ equivocating\text{-}validators\text{-}paper\ \sigma$
  by (*smt Collect-cong Params.equivocating-validators-paper-def equivocating-validators-def is-equivocating-def mem-Collect-eq observed-type-for-state observed-def subsetCE*)

lemma (**in** *Protocol*) *equivocation-is-monotonic* :
  $\forall\ \sigma\ \sigma'\ v.\ is\text{-}future\text{-}state\ (\sigma,\ \sigma')\ \wedge\ v\ \in\ V$
  $\longrightarrow v\ \in\ equivocating\text{-}validators\ \sigma$
  $\longrightarrow v\ \in\ equivocating\text{-}validators\ \sigma'$
  apply (*simp add*: *equivocating-validators-def is-equivocating-def*)
  using *observed-def* **by** *fastforce*

lemma (**in** *Protocol*) *equivocating-validators-preserved-over-honest-message* :
  $\forall\ \sigma\ m.\ \sigma \in \Sigma \wedge m \in M$
  $\longrightarrow sender\ m\ \notin\ equivocating\text{-}validators\ (\sigma\ \cup\ \{m\})$
  $\longrightarrow equivocating\text{-}validators\ \sigma\ =\ equivocating\text{-}validators\ (\sigma\ \cup\ \{m\})$

apply (*simp add*: *equivocating-validators-def is-equivocating-def observed-def equivocation-def*)
by *auto*

definition (**in** *Params*) *weight-measure* :: *validator set* ⇒ *real*
where

weight-measure v-set = *sum W v-set*

lemma (**in** *Params*) *weight-measure-subset-minus* :
finite A ⟹ *finite B* ⟹ *A* ⊆ *B*
⟹ *weight-measure B* − *weight-measure A* = *weight-measure* (*B* − *A*)
apply (*simp add*: *weight-measure-def*)
by (*simp add*: *sum-diff*)

lemma (**in** *Params*) *weight-measure-strict-subset-minus* :
finite A ⟹ *finite B* ⟹ *A* ⊂ *B*
⟹ *weight-measure B* − *weight-measure A* = *weight-measure* (*B* − *A*)
apply (*simp add*: *weight-measure-def*)
by (*simp add*: *sum-diff*)

lemma (**in** *Params*) *weight-measure-disjoint-plus* :
finite A ⟹ *finite B* ⟹ *A* ∩ *B* = ∅
⟹ *weight-measure A* + *weight-measure B* = *weight-measure* (*A* ∪ *B*)
apply (*simp add*: *weight-measure-def*)
by (*simp add*: *sum.union-disjoint*)

lemma (**in** *Protocol*) *weight-positive* :
A ⊆ *V* ⟹ *weight-measure A* ≥ *0*
apply (*simp add*: *weight-measure-def*)
using *W-type*
by (*smt subsetCE sum-nonneg*)

lemma (**in** *Protocol*) *weight-gte-diff* :
A ⊆ *V* ⟹ *weight-measure B* ≥ *weight-measure B* − *weight-measure A*
using *weight-positive* **by** *auto*

lemma (**in** *Protocol*) *weight-measure-subset-gte-diff* :
A ⊆ *V* ⟹ *A* ⊆ *B* ⟹ *weight-measure B* ≥ *weight-measure* (*B* − *A*)
using *weight-measure-subset-minus V-type weight-gte-diff*
by (*smt finite-Diff2 finite-subset sum.infinite weight-measure-def*)

lemma (**in** *Protocol*) *weight-measure-subset-gte* :
B ⊆ *V* ⟹ *A* ⊆ *B* ⟹ *weight-measure B* ≥ *weight-measure A*
using *W-type V-type*

apply (*simp add*: *weight-measure-def*)
by (*smt DiffD1 Params.weight-measure-def finite-subset subsetCE sum-nonneg weight-measure-subset-minus*)

lemma (**in** *Protocol*) *weight-measure-stritct-subset-gt* :
$B \subseteq V \Longrightarrow A \subset B \Longrightarrow$ *weight-measure* $B >$ *weight-measure* A
proof −
fix A B
assume $B \subseteq V$
and $A \subset B$
then have $A \subset V$
by *auto*
have *finite* $A \wedge$ *finite* B
using *V-type finite-subset* ‹$B \subseteq V$› ‹$A \subset B$› **by** *auto*
have $B - A \neq \emptyset \wedge B - A \subseteq V$
using ‹$A \subset B$› ‹$B \subseteq V$›
by *blast*
then have *weight-measure* $(B - A) > 0$
using *W-type*
apply (*simp add*: *weight-measure-def*)
by (*meson Diff-eq-empty-iff V-type rev-finite-subset subset-eq sum-pos*)
have *weight-measure* $B =$ *weight-measure* $(B - A) +$ *weight-measure* A
using *weight-measure-strict-subset-minus* ‹$B \subseteq V$› ‹$A \subset B$› ‹*finite* $A \wedge$ *finite* B›
by *fastforce*
then show *weight-measure* $B >$ *weight-measure* A
using ‹*weight-measure* $(B - A) > 0$›
by *linarith*
qed

definition (**in** *Params*) *equivocation-fault-weight* :: *state* ⇒ *real*
where

equivocation-fault-weight $\sigma =$ *weight-measure* (*equivocating-validators* σ)

lemma (**in** *Protocol*) *equivocation-fault-weight-is-monotonic* :
$\forall\ \sigma\ \sigma'.\ \sigma \in \Sigma \wedge \sigma' \in \Sigma \wedge$ *is-future-state* (σ, σ')
$\longrightarrow$ *equivocation-fault-weight* $\sigma \leq$ *equivocation-fault-weight* σ'
using *equivocation-is-monotonic weight-measure-subset-gte*
by (*smt equivocating-validators-is-finite equivocating-validators-type equivocation-fault-weight-def subset-iff*)

definition (**in** *Params*) *is-faults-lt-threshold* :: *state* ⇒ *bool*

where
is-faults-lt-threshold σ = (*equivocation-fault-weight* $\sigma < t$)

definition (**in** *Protocol*) Σt :: *state set*
where
$\Sigma t = \{\sigma \in \Sigma.$ *is-faults-lt-threshold* $\sigma\}$

lemma (**in** *Protocol*) Σ*t-is-subset-of-*Σ : $\Sigma t \subseteq \Sigma$
using Σ*t-def* **by** *auto*

lemma (**in** *Protocol*) *past-state-of-*Σ*t-is-*Σ*t* :
$\forall\ \sigma\ \sigma'.\ \sigma' \in \Sigma t \wedge \sigma \in \Sigma \wedge$ *is-future-state* (σ, σ')
$\longrightarrow \sigma \in \Sigma t$
using *equivocation-fault-weight-is-monotonic*
apply (*simp add*: Σ*t-def is-faults-lt-threshold-def*)
by *fastforce*

definition (**in** *Protocol*) *futures* :: *state* $\Rightarrow$ *state set*
where
futures $\sigma = \{\sigma' \in \Sigma t.$ *is-future-state* $(\sigma, \sigma')\}$

type-synonym *state-property* = *state* $\Rightarrow$ *bool*

type-synonym *consensus-value-property* = *consensus-value* $\Rightarrow$ *bool*

end

2 Message Justification

theory *MessageJustification*

imports *Main CBCCasper Libraries/LaTeXsugar*

begin

definition (**in** *Params*) *message-justification* :: *message rel*
where
message-justification = $\{(m1, m2).\ \{m1, m2\} \subseteq M \wedge$ *justified m1 m2*$\}$

lemma (**in** *Protocol*) *transitivity-of-justifications* :
trans message-justification
apply (*simp add*: *trans-def message-justification-def justified-def*)

```
  by (meson Params.M-type Params.state-is-in-pow-Mi Protocol-axioms contra-subsetD)

lemma (in Protocol) irreflexivity-of-justifications :
  irrefl message-justification
  apply (simp add: irrefl-def message-justification-def justified-def)
  apply (simp add: M-def)
  apply auto
proof −
  fix n m
  assume est m ∈ C
  assume sender m ∈ V
  assume justification m ∈ Σi (V, C, ε) n
  assume est m ∈ ε (justification m)
  assume m ∈ justification m
  have m ∈ Mi (V, C, ε) (n − 1)
    by (smt Mi.simps One-nat-def Params.Σi-subset-Mi Pow-iff Suc-pred ‹est m ∈
C› ‹est m ∈ ε (justification m)› ‹justification m ∈ Σi (V, C, ε) n› ‹m ∈ justification
m› ‹sender m ∈ V› add.right-neutral add-Suc-right diff-is-0-eq' diff-le-self diff-zero
mem-Collect-eq not-gr0 subsetCE)
  then have justification m ∈ Σi (V, C, ε) (n − 1)
    using Mi.simps by blast
  then have justification m ∈ Σi (V, C, ε) 0
    apply (induction n)
    apply simp
     by (smt Mi.simps One-nat-def Params.Σi-subset-Mi Pow-iff Suc-pred ‹m ∈
justification m› add.right-neutral add-Suc-right diff-Suc-1 mem-Collect-eq not-gr0
subsetCE subsetCE)
  then have justification m ∈ {∅}
    by simp
  then show False
    using ‹m ∈ justification m› by blast
qed

lemma (in Protocol) message-cannot-justify-itself :
  (∀ m ∈ M. ¬ justified m m)
proof −
  have irrefl message-justification
    using irreflexivity-of-justifications by simp
  then show ?thesis
    by (simp add: irreflexivity-of-justifications irrefl-def message-justification-def)
qed

lemma (in Protocol) justification-is-strict-partial-order-on-M :
  strict-partial-order message-justification
  apply (simp add: strict-partial-order-def)
  by (simp add: irreflexivity-of-justifications transitivity-of-justifications)

lemma (in Protocol) monotonicity-of-justifications :
  ∀ m m' σ. m ∈ M ∧ σ ∈ Σ ∧ justified m' m ⟶ justification m' ⊆ justification
```

```
m
  apply simp
  by (meson M-type justified-def message-in-state-is-valid state-is-in-pow-Mi)

lemma (in Protocol) strict-monotonicity-of-justifications :
  ∀ m m′ σ. m ∈ M ∧ σ ∈ Σ ∧ justified m′ m ⟶ justification m′ ⊂ justification
m
  by (metis M-type message-cannot-justify-itself justified-def message-in-state-is-valid
monotonicity-of-justifications psubsetI)

lemma (in Protocol) justification-implies-different-messages :
  ∀ m m′. m ∈ M ∧ m′ ∈ M ⟶ justified m′ m ⟶ m ≠ m′
  using message-cannot-justify-itself by auto

lemma (in Protocol) only-valid-message-is-justified :
  ∀ m ∈ M. ∀ m′. justified m′ m ⟶ m′ ∈ M
  apply (simp add: justified-def)
  using Params.M-type message-in-state-is-valid by blast

lemma (in Protocol) justified-message-exists-in-Mi-n-minus-1 :
  ∀ n m m′. n ∈ ℕ
  ⟶ justified m′ m
  ⟶ m ∈ Mi (V, C, ε) n
  ⟶  m′ ∈ Mi (V, C, ε) (n − 1)
proof −
  have ∀ n m m′. justified m′ m
  ⟶ m ∈ Mi (V, C, ε) n
  ⟶ m ∈ M ∧ m′ ∈ M
  ⟶ m′ ∈ Mi (V, C, ε) (n − 1)
    apply (rule, rule, rule, rule, rule, rule)
  proof −
    fix n m m′
    assume justified m′ m
    assume m ∈ Mi (V, C, ε) n
    assume m ∈ M ∧ m′ ∈ M
    then have justification m ∈ Σi (V,C,ε) n
      using Mi.simps ‹m ∈ Mi (V, C, ε) n› by blast
    then have justification m ∈ Pow (Mi (V,C,ε) (n − 1))
      by (metis (no-types, lifting) Suc-diff-Suc Σi.simps(1) Σi-subset-Mi ‹justified
m′ m› add-leE diff-add diff-le-self empty-iff justified-def neq0-conv plus-1-eq-Suc
singletonD subsetCE)
    show m′ ∈ Mi (V, C, ε) (n − 1)
        using ‹justification m ∈ Pow (Mi (V, C, ε) (n − 1))› ‹justified m′ m›
justified-def by auto
  qed
  then show ?thesis
    by (metis (no-types, lifting) M-def UN-I only-valid-message-is-justified)
qed
```

```
lemma (in Protocol) monotonicity-of-card-of-justification :
  ∀ m m′. m ∈ M
  ⟶ justified m′ m
  ⟶ card (justification m′) < card (justification m)
  by (meson M-type Protocol.strict-monotonicity-of-justifications Protocol-axioms
justification-is-finite psubset-card-mono)


lemma (in Protocol) justification-is-well-founded-on-M :
  wfp-on justified M
proof (rule ccontr)
  assume ¬ wfp-on justified M
  then have ∃f. ∀i. f i ∈ M ∧ justified (f (Suc i)) (f i)
    by (simp add: wfp-on-def)
  then obtain f where ∀i. f i ∈ M ∧ justified (f (Suc i)) (f i) by auto
  have ∀ i. card (justification (f i)) ≤ card (justification (f 0)) − i
    apply (rule)
  proof −
    fix i
    have card (justification (f (Suc i))) < card (justification (f i))
   using ‹∀i. f i ∈ M ∧ justified (f (Suc i)) (f i)› by (simp add: monotonicity-of-card-of-justification)
    show card (justification (f i)) ≤ card (justification (f 0)) − i
      apply (induction i)
      apply simp
      using ‹card (justification (f (Suc i))) < card (justification (f i))›
       by (smt Suc-diff-le ‹∀i. f i ∈ M ∧ justified (f (Suc i)) (f i)› diff-Suc-Suc
diff-is-0-eq le-iff-add less-Suc-eq-le less-imp-le monotonicity-of-card-of-justification
not-less-eq-eq trans-less-add1)
  qed
  then have ∃ i. i = card (justification (f 0)) + Suc 0 ∧ card (justification (f i))
≤ card (justification (f 0)) − i
    by blast
  then show False
     using le-0-eq le-simps(2) linorder-not-le monotonicity-of-card-of-justification
nat-diff-split order-less-imp-le
   by (metis ‹∀i. f i ∈ M ∧ justified (f (Suc i)) (f i)› add.right-neutral add-Suc-right)
qed

lemma (in Protocol) subset-of-M-have-minimal-of-justification :
  ∀ S ⊆ M. S ≠ ∅ ⟶ (∃ m-min ∈ S. ∀ m. justified m m-min ⟶ m ∉ S)
 by (metis justification-is-well-founded-on-M wfp-on-imp-has-min-elt wfp-on-mono)

lemma (in Protocol) message-in-state-is-strict-subset-of-the-state :
  ∀ σ ∈ Σ. ∀ m ∈ σ. justification m ⊂ σ
 using justification-implies-different-messages justified-def message-in-state-is-valid
state-is-in-pow-Mi by fastforce

end
```

3 Latest Message

theory *LatestMessage*

imports *Main CBCCasper MessageJustification Libraries/LaTeXsugar*

begin

definition *later* :: (*message* $*$ *message set*) $\Rightarrow$ *message set*
  where
    later = (λ(*m*, σ). {*m'* $\in$ σ. *justified m m'*})

lemma (**in** *Protocol*) *later-type* :
  $\forall$ σ *m*. σ $\in$ *Pow M* $\wedge$ *m* $\in$ *M* $\longrightarrow$ *later* (*m*, σ) $\subseteq$ *M*
  apply (*simp add*: *later-def*)
  by *auto*

lemma (**in** *Protocol*) *later-type-for-state* :
  $\forall$ σ *m*. σ $\in$ Σ $\wedge$ *m* $\in$ *M* $\longrightarrow$ *later* (*m*, σ) $\subseteq$ *M*
  apply (*simp add*: *later-def*)
  using *state-is-subset-of-M* **by** *auto*

definition *from-sender* :: (*validator* $*$ *message set*) $\Rightarrow$ *message set*
  where
    from-sender = (λ(*v*, σ). {*m* $\in$ σ. *sender m* = *v*})

lemma (**in** *Protocol*) *from-sender-type* :
  $\forall$ σ *v*. σ $\in$ *Pow M* $\wedge$ *v* $\in$ *V* $\longrightarrow$ *from-sender* (*v*, σ) $\in$ *Pow M*
  apply (*simp add*: *from-sender-def*)
  by *auto*

lemma (**in** *Protocol*) *from-sender-type-for-state* :
  $\forall$ σ *v*. σ $\in$ Σ $\wedge$ *v* $\in$ *V* $\longrightarrow$ *from-sender* (*v*, σ) $\subseteq$ *M*
  apply (*simp add*: *from-sender-def*)
  using *state-is-subset-of-M* **by** *auto*

lemma (**in** *Protocol*) *messages-from-observed-validator-is-non-empty* :
  $\forall$ σ *v*. σ $\in$ Σ $\wedge$ *v* $\in$ *observed* σ $\longrightarrow$ *from-sender* (*v*, σ) $\neq$ $\emptyset$
  apply (*simp add*: *observed-def from-sender-def*)
  by *auto*

lemma (**in** *Protocol*) *messages-from-validator-is-finite* :
  $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V\sigma \longrightarrow$ *finite* (*from-sender* (v, σ))
  by (*simp add*: *from-sender-def state-is-finite*)

definition *from-group* :: (*validator set* $*$ *message set*) $\Rightarrow$ *state*
  where
    from-group $= (\lambda(v\text{-}set, \sigma).\ \{m \in \sigma.\ sender\ m \in v\text{-}set\})$

lemma (**in** *Protocol*) *from-group-type* :
  $\forall\ \sigma\ v.\ \sigma \in Pow\ M \wedge v\text{-}set \subseteq V \longrightarrow$ *from-group* $(v\text{-}set, \sigma) \in Pow\ M$
  apply (*simp add*: *from-group-def*)
  by *auto*

lemma (**in** *Protocol*) *from-group-type-for-state* :
  $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v\text{-}set \subseteq V \longrightarrow$ *from-group* $(v\text{-}set, \sigma) \subseteq M$
  apply (*simp add*: *from-group-def*)
  using *state-is-subset-of-M* **by** *auto*

definition *later-from* :: (*message* $*$ *validator* $*$ *message set*) $\Rightarrow$ *message set*
  where
    later-from $= (\lambda(m, v, \sigma).\ \{m' \in \sigma.\ sender\ m' = v \wedge justified\ m\ m'\})$

lemma (**in** *Protocol*) *later-from-type* :
  $\forall\ \sigma\ v\ m.\ \sigma \in Pow\ M \wedge v \in V \wedge m \in M \longrightarrow$ *later-from* $(m, v, \sigma) \in Pow\ M$
  apply (*simp add*: *later-from-def*)
  by *auto*

lemma (**in** *Protocol*) *later-from-type-for-state* :
  $\forall\ \sigma\ v\ m.\ \sigma \in \Sigma \wedge v \in V \wedge m \in M \longrightarrow$ *later-from* $(m, v, \sigma) \subseteq M$
  apply (*simp add*: *later-from-def*)
  using *message-in-state-is-valid* **by** *auto*

definition *L-M* :: *message set* $\Rightarrow$ (*validator* $\Rightarrow$ *message set*)
  where
    L-M $\sigma\ v = \{m \in$ *from-sender* (v, σ). *later-from* $(m, v, \sigma) = \emptyset\}$

lemma (**in** *Protocol*) *L-M-type* :
  $\forall\ \sigma\ v.\ \sigma \in Pow\ M \wedge v \in V \longrightarrow$ *L-M* $\sigma\ v \in Pow\ M$
  apply (*simp add*: *L-M-def later-from-def*)
  using *from-sender-type* **by** *auto*

lemma (**in** *Protocol*) *L-M-type-for-state* :
  $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *L-M* $\sigma\ v \subseteq M$
  apply (*simp add*: *L-M-def later-from-def*)

using *from-sender-type-for-state* **by** *auto*

lemma (**in** *Protocol*) *L-M-from-non-observed-validator-is-empty* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \wedge v \notin$ *observed* $\sigma \longrightarrow$ *L-M* $\sigma\ v = \emptyset$
by (*simp add*: *L-M-def observed-def later-def from-sender-def*)

lemma (**in** *Protocol*) *L-M-is-subset-of-the-state* :
$\forall\ \sigma \in \Sigma.\ \forall\ v \in V.$ *L-M* $\sigma\ v \subseteq \sigma$
by (*simp add*: *L-M-def later-from-def from-sender-def*)

definition *observed-non-equivocating-validators* :: *state* $\Rightarrow$ *validator set*
where
observed-non-equivocating-validators σ = *observed* σ − *equivocating-validators* σ

lemma (**in** *Protocol*) *observed-non-equivocating-validators-type* :
$\forall\ \sigma \in \Sigma.$ *observed-non-equivocating-validators* $\sigma \in$ *Pow V*
apply (*simp add*: *observed-non-equivocating-validators-def*)
using *observed-type-for-state equivocating-validators-type* **by** *auto*

lemma (**in** *Protocol*) *observed-non-equivocating-validators-are-not-equivocating* :
$\forall\ \sigma \in \Sigma.$ *observed-non-equivocating-validators* $\sigma \cap$ *equivocating-validators* $\sigma = \emptyset$
unfolding *observed-non-equivocating-validators-def*
by *blast*

lemma (**in** *Protocol*) *justification-is-well-founded-on-messages-from-validator*:
$\forall\ \sigma \in \Sigma.\ (\forall\ v \in V.$ *wfp-on justified* (*from-sender* (v, σ)))
using *justification-is-well-founded-on-M from-sender-type-for-state wfp-on-subset*
by *blast*

lemma (**in** *Protocol*) *justification-is-total-on-messages-from-non-equivocating-validator*:
$\forall\ \sigma \in \Sigma.\ (\forall\ v \in V.\ v \notin$ *equivocating-validators* $\sigma \longrightarrow$ *Relation.total-on* (*from-sender* (v, σ)) *message-justification*)
proof −
have $\forall$ *m1 m2* $\sigma\ v.\ v \in V \wedge \sigma \in \Sigma \wedge \{m1, m2\} \subseteq$ *from-sender* $(v, \sigma) \longrightarrow$ *sender m1* = *sender m2*
by (*simp add*: *from-sender-def*)
then have $\forall\ \sigma \in \Sigma.\ (\forall\ v \in V.\ v \notin$ *equivocating-validators* σ
$\longrightarrow$ ($\forall$ *m1 m2*. $\{m1, m2\} \subseteq$ *from-sender* $(v, \sigma) \longrightarrow m1 = m2 \vee$ *justified m1 m2* $\vee$ *justified m2 m1*))
apply (*simp add*: *equivocating-validators-def is-equivocating-def equivocation-def from-sender-def observed-def*)
by *blast*
then show *?thesis*
apply (*simp add*: *Relation.total-on-def message-justification-def*)
using *from-sender-type-for-state* **by** *blast*
qed

lemma (**in** *Protocol*) *justification-is-strict-linear-order-on-messages-from-non-equivocating-validator*:
  $\forall\ \sigma \in \Sigma.$ ($\forall\ v \in V.$ $v \notin$ *equivocating-validators* $\sigma \longrightarrow$ *strict-linear-order-on*
(*from-sender* (v, σ)) *message-justification*)
 by (*simp add*: *strict-linear-order-on-def justification-is-total-on-messages-from-non-equivocating-validator*

    irreflexivity-of-justifications transitivity-of-justifications)

lemma (**in** *Protocol*) *justification-is-strict-well-order-on-messages-from-non-equivocating-validator*:
  $\forall\ \sigma \in \Sigma.$ ($\forall\ v \in V.$ $v \notin$ *equivocating-validators* σ
  $\longrightarrow$ *strict-linear-order-on* (*from-sender* (v, σ)) *message-justification* $\wedge$ *wfp-on*
justified (*from-sender* (v, σ)))
  using *justification-is-well-founded-on-messages-from-validator*
    justification-is-strict-linear-order-on-messages-from-non-equivocating-validator

  by *blast*

lemma (**in** *Protocol*) *latest-message-is-maximal-element-of-justification* :
  $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *L-M* σ v = {m. *maximal-on* (*from-sender* (v, σ))
message-justification m}
 apply (*simp add*: *L-M-def later-from-def from-sender-def message-justification-def*
maximal-on-def)
  using *from-sender-type-for-state* **apply** *auto*
  using *message-in-state-is-valid* **by** *blast*

lemma (**in** *Protocol*) *observed-non-equivocating-validators-have-one-latest-message*:
  $\forall\ \sigma \in \Sigma.$ ($\forall\ v \in$ *observed-non-equivocating-validators* σ. *is-singleton* (*L-M* σ v))

  apply (*simp add*: *observed-non-equivocating-validators-def*)
proof −
  have $\forall\ \sigma \in \Sigma.$ ($\forall\ v \in$ *observed* σ − *equivocating-validators* σ. *is-singleton* {m.
maximal-on (*from-sender* (v, σ)) *message-justification* m})
    using
      messages-from-observed-validator-is-non-empty
      messages-from-validator-is-finite
      observed-type-for-state
      equivocating-validators-def
    justification-is-strict-linear-order-on-messages-from-non-equivocating-validator
      strict-linear-order-on-finite-non-empty-set-has-one-maximum
      maximal-and-maximum-coincide-for-strict-linear-order
    by (*smt Collect-cong DiffD1 DiffD2 set-mp*)
 then show $\forall\sigma\in\Sigma.\ \forall v\in$*observed* σ − *equivocating-validators* σ. *is-singleton* (*L-M*
σ v)
    using *latest-message-is-maximal-element-of-justification*
      observed-non-equivocating-validators-def observed-non-equivocating-validators-type

    by *fastforce*
qed

definition *L-E* :: *state* ⇒ *validator* ⇒ *consensus-value set*
  where
    L-E σ *v* = {*est m* | *m*. *m* ∈ *L-M* σ *v*}

lemma (**in** *Protocol*) *L-E-type* :
  ∀ σ *v*. $\sigma \in \Sigma \land v \in V \longrightarrow$ *L-E* σ *v* ⊆ *C*
  using *M-type Protocol.L-M-type-for-state Protocol-axioms L-E-def* **by** *fastforce*

lemma (**in** *Protocol*) *L-E-from-non-observed-validator-is-empty* :
  ∀ σ *v*. $\sigma \in \Sigma \land v \in V \land v \notin$ *observed* $\sigma \longrightarrow$ *L-E* σ *v* = ∅
  using *L-E-def L-M-from-non-observed-validator-is-empty* **by** *auto*

definition *L-H-M* :: *state* ⇒ *validator* ⇒ *message set*
  where
    L-H-M σ *v* = (*if v* ∈ *equivocating-validators* σ *then* ∅ *else L-M* σ *v*)

lemma (**in** *Protocol*) *L-H-M-type* :
  ∀ σ *v*. $\sigma \in \Sigma \land v \in V \longrightarrow$ *L-H-M* σ *v* ⊆ *M*
  by (*simp add*: *L-M-type-for-state L-H-M-def*)

lemma (**in** *Protocol*) *L-H-M-of-observed-non-equivocating-validator-is-singleton* :
  ∀ $\sigma \in \Sigma$. ∀ *v* ∈ *observed-non-equivocating-validators* σ.
    is-singleton (*L-H-M* σ *v*)
  using *observed-non-equivocating-validators-have-one-latest-message*
  by (*simp add*: *L-H-M-def observed-non-equivocating-validators-def*)

lemma (**in** *Protocol*) *sender-of-L-H-M*:
  ∀ $\sigma \in \Sigma$. ∀ *v* ∈ *observed-non-equivocating-validators* σ. *sender* (*the-elem* (*L-H-M*
σ *v*)) = *v*
    using *L-H-M-of-observed-non-equivocating-validator-is-singleton*
      L-H-M-def L-M-def from-sender-def
   by (*smt Diff-iff is-singleton-the-elem mem-Collect-eq observed-non-equivocating-validators-def*
prod.simps(*2*) *singletonI*)

lemma (**in** *Protocol*) *L-H-M-is-in-the-state*:
  ∀ $\sigma \in \Sigma$. ∀ *v* ∈ *observed-non-equivocating-validators* σ. *the-elem* (*L-H-M* σ *v*)

$\in \sigma$
  using *L-H-M-of-observed-non-equivocating-validator-is-singleton*
    L-H-M-def L-M-is-subset-of-the-state
 by (*metis Diff-iff contra-subsetD insert-subset is-singleton-the-elem observed-non-equivocating-validators-def observed-type-for-state*)

definition *L-H-E* :: *state* ⇒ *validator* ⇒ *consensus-value set*
 where

  $L\text{-}H\text{-}E\ \sigma\ v = est\ \text{`}L\text{-}H\text{-}M\ \sigma\ v$

lemma (**in** *Protocol*) *L-H-E-type* :
 $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow L\text{-}H\text{-}E\ \sigma\ v \in Pow\ C$
 using *Protocol.L-E-type Protocol-axioms L-E-def L-H-E-def L-H-M-def*
 using *M-type L-H-M-type* **by** *fastforce*

lemma (**in** *Protocol*) *L-H-E-from-non-observed-validator-is-empty* :
 $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \wedge v \notin observed\ \sigma \longrightarrow L\text{-}H\text{-}E\ \sigma\ v = \emptyset$
 by (*simp add*: *L-H-E-def L-H-M-def L-M-from-non-observed-validator-is-empty*)

lemma *image-of-singleton-is-singleton* :
 $is\text{-}singleton\ A \Longrightarrow is\text{-}singleton\ (f\ \text{`}A)$
 apply (*simp add*: *is-singleton-def*)
 by *blast*

lemma (**in** *Protocol*) *L-H-E-of-observed-non-equivocating-validator-is-singleton* :
 $\forall\ \sigma \in \Sigma.\ \forall\ v \in observed\text{-}non\text{-}equivocating\text{-}validators\ \sigma.$
   $is\text{-}singleton\ (L\text{-}H\text{-}E\ \sigma\ v)$
 using *L-H-M-of-observed-non-equivocating-validator-is-singleton*
 apply (*simp add*: *L-H-E-def*)
 using *image-of-singleton-is-singleton*
 by *blast*

definition *L-H-J* :: *state* ⇒ *validator* ⇒ *state set*
 where
  $L\text{-}H\text{-}J\ \sigma\ v = justification\ \text{`}L\text{-}H\text{-}M\ \sigma\ v$

lemma (**in** *Protocol*) *L-H-J-type* :
 $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow L\text{-}H\text{-}J\ \sigma\ v \subseteq \Sigma$
 using *M-type L-H-M-type*

```
    L-H-J-def by auto

lemma (in Protocol) L-H-J-of-observed-non-equivocating-validator-is-singleton :
  ∀ σ ∈ Σ. v ∈ observed-non-equivocating-validators σ
    ⟶ is-singleton (L-H-J σ v)
  using L-H-M-of-observed-non-equivocating-validator-is-singleton
  apply (simp add: L-H-J-def)
  using image-of-singleton-is-singleton
  by blast

lemma (in Protocol) L-H-J-is-subset-of-the-state :
  ∀ σ v. σ ∈ Σ ∧ v ∈ V ⟶ (∀ σ' ∈ L-H-J σ v. σ' ⊂ σ)
  apply (simp add: L-H-J-def
                   L-H-M-def)
  using L-M-is-subset-of-the-state
      message-in-state-is-strict-subset-of-the-state
  by blast


end
theory StateTransition

imports Main CBCCasper MessageJustification

begin





definition (in Params) state-transition :: state rel
  where
    state-transition = {(σ1, σ2). {σ1, σ2} ⊆ Σ ∧ is-future-state(σ1, σ2)}

lemma (in Params) reflexivity-of-state-transition :
  refl-on Σ state-transition
  apply (simp add: state-transition-def refl-on-def)
  by auto

lemma (in Params) transitivity-of-state-transition :
  trans state-transition
  apply (simp add: state-transition-def trans-def)
  by auto

lemma (in Params) state-transition-is-preorder :
  preorder-on Σ state-transition
 by (simp add: preorder-on-def reflexivity-of-state-transition transitivity-of-state-transition)

lemma (in Params) antisymmetry-of-state-transition :
```

antisym state-transition
apply (*simp add*: *state-transition-def antisym-def*)
by *auto*

lemma (**in** *Params*) *state-transition-is-partial-order* :
partial-order-on Σ *state-transition*
by (*simp add*: *partial-order-on-def state-transition-is-preorder antisymmetry-of-state-transition*)

definition *immediately-next-message* **where**
immediately-next-message = ($\lambda(\sigma, m)$. *justification* $m \subseteq \sigma \wedge m \notin \sigma$)

lemma (**in** *Protocol*) *state-transition-by-immediately-next-message-of-same-depth-non-zero*:

$\forall n \geq 1. \forall \sigma \in \Sigma i\ (V,C,\varepsilon)\ n. \forall m \in Mi\ (V,C,\varepsilon)\ n.$ *immediately-next-message* (σ,m)
$\longrightarrow \sigma \cup \{m\} \in \Sigma i\ (V,C,\varepsilon)\ (n+1)$
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof−
fix $n\ \sigma\ m$
assume $1 \leq n\ \sigma \in \Sigma i\ (V, C, \varepsilon)\ n\ m \in Mi\ (V, C, \varepsilon)\ n$ *immediately-next-message*
(σ, m)

have $\exists n'. n = Suc\ n'$
using ‹$1 \leq n$› *old.nat.exhaust* **by** *auto*
hence *si*: $\Sigma i\ (V,C,\varepsilon)\ n = \{\sigma \in Pow\ (Mi\ (V,C,\varepsilon)\ (n - 1))$. *finite* $\sigma \wedge (\forall\ m.$
$m \in \sigma \longrightarrow$ *justification* $m \subseteq \sigma)\}$
by *force*

hence $\Sigma i\ (V,C,\varepsilon)\ (n+1) = \{\sigma \in Pow\ (Mi\ (V,C,\varepsilon)\ n)$. *finite* $\sigma \wedge (\forall\ m.\ m \in$
$\sigma \longrightarrow$ *justification* $m \subseteq \sigma)\}$
by *force*

have *justification* $m \subseteq \sigma$
using *immediately-next-message-def*
by (*metis* (*no-types*, *lifting*) ‹*immediately-next-message* (σ, m)› *case-prod-conv*)
hence *justification* $m \subseteq \sigma \cup \{m\}$
by *blast*
moreover have $\bigwedge m'$. *finite* $\sigma \wedge m' \in \sigma \Longrightarrow$ *justification* $m' \subseteq \sigma$
using ‹$\sigma \in \Sigma i\ (V, C, \varepsilon)\ n$› *si* **by** *blast*
hence$\bigwedge m'$. *finite* $\sigma \wedge m' \in \sigma \Longrightarrow$ *justification* $m' \subseteq \sigma \cup \{m\}$
by *auto*
ultimately have $\bigwedge m'. m' \in \sigma \cup \{m\} \Longrightarrow$ *justification* $m \subseteq \sigma$
using ‹*justification* $m \subseteq \sigma$› **by** *blast*

have $\{m\} \in Pow\ (Mi\ (V,C,\varepsilon)\ n)$
using ‹$m \in Mi\ (V, C, \varepsilon)\ n$› **by** *auto*
moreover have $\sigma \in Pow\ (Mi\ (V,C,\varepsilon)\ (n-1))$
using ‹$\sigma \in \Sigma i\ (V, C, \varepsilon)\ n$› *si* **by** *auto*
hence $\sigma \in Pow\ (Mi\ (V,C,\varepsilon)\ n)$

using *Mi-monotonic*
by (*metis* (*full-types*) *PowD PowI Suc-eq-plus1* ‹∃ *n*′. *n* = *Suc n*′› *diff-Suc-1 subset-iff*)
ultimately have σ ∪ {*m*} ∈ *Pow* (*Mi* (*V*,*C*,ε) *n*)
by *blast*

show σ ∪ {*m*} ∈ Σ*i* (*V*, *C*, ε) (*n* + *1*)
using ‹⋀*m*′. *finite* σ ∧ *m*′ ∈ σ ⟹ *justification m*′ ⊆ σ ∪ {*m*}› ‹σ ∪ {*m*} ∈ *Pow* (*Mi* (*V*, *C*, ε) *n*)› ‹*justification m* ⊆ σ ∪ {*m*}›
‹σ ∈ Σ*i* (*V*, *C*, ε) *n*› *si* **by** *auto*
qed

lemma (**in** *Protocol*) *state-transition-by-immediately-next-message-of-same-depth*:

∀σ∈Σ*i* (*V*,*C*,ε) *n*. ∀*m*∈*Mi* (*V*,*C*,ε) *n*. *immediately-next-message* (σ,*m*) ⟶ σ ∪ {*m*} ∈ Σ*i* (*V*,*C*,ε) (*n*+*1*)
apply (*cases n*)
apply *auto*[*1*]
using *state-transition-by-immediately-next-message-of-same-depth-non-zero*
by (*metis le-add1 plus-1-eq-Suc*)

lemma (**in** *Params*) *past-state-exists-in-same-depth* :
∀ σ σ′. σ′ ∈ Σ*i* (*V*,*C*,ε) *n* ⟶ σ ⊆ σ′ ⟶ σ ∈ Σ ⟶ σ ∈ Σ*i* (*V*,*C*,ε) *n*
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof (*cases n*)
case *0*
show ⋀σ σ′. σ′ ∈ Σ*i* (*V*, *C*, ε) *n* ⟹ σ ⊆ σ′ ⟹ σ ∈ Σ ⟹ *n* = *0* ⟹ σ ∈ Σ*i* (*V*, *C*, ε) *n*
by *auto*
next
case (*Suc nat*)
show ⋀σ σ′ *nat*. σ′ ∈ Σ*i* (*V*, *C*, ε) *n* ⟹ σ ⊆ σ′ ⟹ σ ∈ Σ ⟹ *n* = *Suc nat* ⟹ σ ∈ Σ*i* (*V*, *C*, ε) *n*
proof −
fix σ σ′
assume σ′ ∈ Σ*i* (*V*, *C*, ε) *n*
and σ ⊆ σ′
and σ ∈ Σ
have *n* > *0*
by (*simp add*: *Suc*)
have *finite* σ ∧ (∀ *m*. *m* ∈ σ ⟶ *justification m* ⊆ σ)
using ‹σ ∈ Σ› *state-is-finite state-is-in-pow-Mi* **by** *blast*
moreover have σ ∈ *Pow* (*Mi* (*V*, *C*, ε) (*n* − *1*))
using ‹σ ⊆ σ′›
by (*smt Pow-iff Suc-eq-plus1* Σ*i-monotonic* Σ*i-subset-Mi* ‹σ′ ∈ Σ*i* (*V*, *C*, ε) *n*› *add-diff-cancel-left*′ *add-eq-if diff-is-0-eq diff-le-self plus-1-eq-Suc subset-iff*)
ultimately have σ ∈ {σ ∈ *Pow* (*Mi* (*V*,*C*,ε) (*n* − *1*)). *finite* σ ∧ (∀ *m*. *m* ∈ σ ⟶ *justification m* ⊆ σ)}
by *blast*

then show $\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$
by (*simp add*: *Suc*)
qed
qed

lemma (**in** *Protocol*) *immediately-next-message-exists-in-same-depth*:
$\forall\ \sigma \in \Sigma.\ \forall\ m \in M.$ *immediately-next-message* $(\sigma,m) \longrightarrow (\exists\ n \in \mathbb{N}.\ \sigma \in \Sigma i$ $(V,C,\varepsilon)\ n \wedge m \in Mi\ (V,C,\varepsilon)\ n)$
apply (*simp add*: *immediately-next-message-def M-def* Σ*-def*)
using *past-state-exists-in-same-depth*
using Σi*-is-subset-of-*Σ **by** *blast*

lemma (**in** *Protocol*) *state-transition-by-immediately-next-message*:
$\forall\ \sigma \in\Sigma.\ \forall\ m \in M.$ *immediately-next-message* $(\sigma,m) \longrightarrow \sigma \cup \{m\} \in \Sigma$
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ m$
assume $\sigma \in \Sigma$
and $m \in M$
and *immediately-next-message* $(\sigma,\ m)$
then have $(\exists\ n \in \mathbb{N}.\ \sigma \in \Sigma i\ (V,C,\varepsilon)\ n \wedge m \in Mi\ (V,C,\varepsilon)\ n)$
using *immediately-next-message-exists-in-same-depth* $\langle\sigma \in \Sigma\rangle$ $\langle m \in M\rangle$
by *blast*
then have $\exists\ n \in \mathbb{N}.\ \sigma \cup \{m\} \in \Sigma i\ (V,C,\varepsilon)\ (n + 1)$
using *state-transition-by-immediately-next-message-of-same-depth*
using ⟨*immediately-next-message* $(\sigma,\ m)$⟩ **by** *blast*
show $\sigma \cup \{m\} \in \Sigma$
apply (*simp add*: Σ*-def*)
by (*metis Nats-1 Nats-add Un-insert-right* $\langle\exists\, n{\in}\mathbb{N}.\ \sigma \cup \{m\} \in \Sigma i\ (V,\ C,\ \varepsilon)$ $(n + 1)\rangle$ *sup-bot.right-neutral*)
qed

lemma (**in** *Protocol*) *state-transition-imps-immediately-next-message*:
$\forall\ \sigma \in\Sigma.\ \forall\ m \in M.\ \sigma \cup \{m\} \in \Sigma \wedge m \notin \sigma \longrightarrow$ *immediately-next-message* (σ,m)
proof −
have $\forall\ \sigma \in\Sigma.\ \forall\ m \in M.\ \sigma \cup \{m\} \in \Sigma \longrightarrow (\forall\ m' \in \sigma \cup \{m\}.$ *justification* m' $\subseteq \sigma \cup \{m\})$
using *state-is-in-pow-Mi* **by** *blast*
then have $\forall\ \sigma \in\Sigma.\ \forall\ m \in M.\ \sigma \cup \{m\} \in \Sigma \longrightarrow$ *justification* $m \subseteq \sigma \cup \{m\}$
by *auto*
then have $\forall\ \sigma \in\Sigma.\ \forall\ m \in M.\ \sigma \cup \{m\} \in \Sigma \wedge m \notin \sigma \longrightarrow$ *justification* $m \subseteq \sigma$
using *justification-implies-different-messages justified-def* **by** *fastforce*
then show *?thesis*
by (*simp add*: *immediately-next-message-def*)
qed

lemma (**in** *Protocol*) *state-transition-only-made-by-immediately-next-message*:
$\forall\ \sigma \in \Sigma.\ \forall\ m \in M.\ \sigma \cup \{m\} \in \Sigma \wedge m \notin \sigma \longleftrightarrow$ *immediately-next-message* $(\sigma,$ $m)$

```
 using state-transition-imps-immediately-next-message state-transition-by-immediately-next-message
  apply (simp add: immediately-next-message-def)
  by blast

lemma (in Protocol) state-transition-is-immediately-next-message:
  $\forall\ \sigma \in \Sigma.\ \forall\ m \in M.\ \sigma \cup \{m\} \in \Sigma\ \longleftrightarrow$ justification m $\subseteq \sigma$
  using state-transition-only-made-by-immediately-next-message
  apply (simp add: immediately-next-message-def)
  using insert-Diff state-is-in-pow-Mi by fastforce


lemma (in Protocol) strict-subset-of-state-have-immediately-next-messages:
  $\forall\ \sigma \in \Sigma.\ \forall\ \sigma'.\ \sigma' \subset \sigma \longrightarrow (\exists\ m \in \sigma - \sigma'.$ immediately-next-message $(\sigma', m))$
  apply (simp add: immediately-next-message-def)
  apply (rule, rule, rule)
proof -
  fix $\sigma$ $\sigma'$
  assume $\sigma \in \Sigma$
  assume $\sigma' \subset \sigma$
  show $\exists\ m \in \sigma - \sigma'.$ justification m $\subseteq \sigma'$
  proof (rule ccontr)
    assume $\neg\ (\exists\ m \in \sigma - \sigma'.$ justification m $\subseteq \sigma')$
    then have $\forall\ m \in \sigma - \sigma'.\ \exists\ m' \in$ justification m. $m' \in \sigma - \sigma'$
      using ‹$\neg\ (\exists m{\in}\sigma - \sigma'.$ justification m $\subseteq \sigma')$› state-is-in-pow-Mi ‹$\sigma' \subset \sigma$›
      by (metis Diff-iff ‹$\sigma \in \Sigma$› subset-eq)
    then have $\forall\ m \in \sigma - \sigma'.\ \exists\ m'.$ justified m' m $\wedge\ m' \in \sigma - \sigma'$
      using justified-def by auto
    then have $\forall\ m \in \sigma - \sigma'.\ \exists\ m'.$ justified m' m $\wedge\ m' \in \sigma - \sigma' \wedge m \neq m'$
     using justification-implies-different-messages  state-difference-is-valid-message
      message-in-state-is-valid  ‹$\sigma' \subset \sigma$›
      by (meson DiffD1 ‹$\sigma \in \Sigma$›)
    have $\sigma - \sigma' \subseteq M$
      using ‹$\sigma \in \Sigma$› ‹$\sigma' \subset \sigma$› state-is-subset-of-M by auto
    then have $\exists$ m-min $\in \sigma - \sigma'.\ \forall$ m. justified m m-min $\longrightarrow m \notin \sigma - \sigma'$
      using subset-of-M-have-minimal-of-justification ‹$\sigma' \subset \sigma$›
      by blast
    then show False
      using ‹$\forall\ m \in \sigma - \sigma'.\ \exists\ m'.$ justified m' m $\wedge\ m' \in \sigma - \sigma'$› by blast
  qed
qed

lemma (in Protocol) intermediate-state-towards-strict-future:
  $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in$ futures $\sigma.\ \sigma \subset \sigma' \longrightarrow (\exists\ m \in \sigma' - \sigma.\ \sigma \cup \{m\} \in \Sigma t)$
  apply (rule, rule, rule)
proof -
  fix $\sigma$ $\sigma'$
  assume $\sigma \in \Sigma$
  assume $\sigma' \in$ futures $\sigma$
  assume $\sigma \subset \sigma'$
```

have $\exists$ *m* $\in \sigma' - \sigma$. *immediately-next-message* (σ, *m*)
using *strict-subset-of-state-have-immediately-next-messages*
‹$\sigma \in \Sigma$› ‹$\sigma \subset \sigma'$› ‹$\sigma' \in$ *futures* σ›
by (*simp add*: *futures-def* Σ*t-def*)
then have $\exists$ *m* $\in \sigma' - \sigma$. $\sigma \cup \{m\} \in \Sigma$
using *state-transition-only-made-by-immediately-next-message* ‹$\sigma \in \Sigma$› ‹$\sigma' \in$ *futures* σ›
by (*smt DiffD1* Σ*t-is-subset-of-*Σ *futures-def mem-Collect-eq message-in-state-is-valid subsetCE*)
then have $\exists$ *m* $\in \sigma' - \sigma$. $\sigma \cup \{m\} \in \Sigma \wedge \sigma \cup \{m\} \subseteq \sigma'$
using ‹$\sigma \subset \sigma'$› **by** *auto*
then show $\exists$ *m* $\in \sigma' - \sigma$. $\sigma \cup \{m\} \in \Sigma t$
using *equivocation-fault-weight-is-monotonic* ‹$\sigma' \in$ *futures* σ›
apply (*simp add*: *futures-def* Σ*t-def is-faults-lt-threshold-def*)
by *fastforce*
qed

lemma (**in** *Protocol*) *intermediate-state-by-immediately-next-message-towards-strict-future*:

$\forall$ $\sigma \in \Sigma t$. $\forall$ $\sigma' \in$ *futures* σ. $\sigma \subset \sigma'$
$\longrightarrow$ ($\exists$ *m* $\in \sigma' - \sigma$. *immediately-next-message* (σ, *m*) $\wedge$ $\sigma \cup \{m\} \in \Sigma t \wedge \sigma' \in$ *futures* ($\sigma \cup \{m\}$))
using *intermediate-state-towards-strict-future*
message-in-state-is-valid state-transition-imps-immediately-next-message
apply (*simp add*: Σ*t-def futures-def*)
by (*meson DiffE*)

lemma (**in** *Protocol*) *state-differences-have-immediately-next-messages*:
$\forall$ $\sigma \in \Sigma$. $\forall$ $\sigma' \in \Sigma$. *is-future-state* (σ, σ') $\wedge$ $\sigma \neq \sigma'$ $\longrightarrow$ ($\exists$ *m* $\in \sigma' - \sigma$. *immediately-next-message* (σ, *m*))
using *strict-subset-of-state-have-immediately-next-messages*
by (*simp add*: *psubsetI*)

lemma (**in** *Protocol*) *union-of-two-states-is-state* :
$\forall$ $\sigma 1 \in \Sigma$. $\forall$ $\sigma 2 \in \Sigma$. ($\sigma 1 \cup \sigma 2$) $\in \Sigma$
apply (*rule*, *rule*)
proof –
fix $\sigma 1$ $\sigma 2$
assume $\sigma 1 \in \Sigma$ **and** $\sigma 2 \in \Sigma$
show $\sigma 1 \cup \sigma 2 \in \Sigma$
proof (*cases* $\sigma 1 \subseteq \sigma 2$)
case *True*
then show *?thesis*
by (*simp add*: *Un-absorb1* ‹$\sigma 2 \in \Sigma$›)
next

case *False*
then have $\neg\ \sigma 1 \subseteq \sigma 2$ **by** *simp*
have $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - (\sigma \cap \sigma').$ *immediately-next-message*$(\sigma \cap \sigma', m))$
by (*metis Int-subset-iff psubsetI strict-subset-of-state-have-immediately-next-messages subsetI*)
then have $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - (\sigma \cap \sigma').$ *immediately-next-message*$(\sigma', m))$
apply (*simp add*: *immediately-next-message-def*)
by *blast*
then have $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)$
using *state-transition-by-immediately-next-message*
by (*metis DiffD1 DiffD2 DiffI IntI message-in-state-is-valid*)
have $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow\ \sigma \cup \sigma' \in \Sigma$
proof −
have $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow$ *card* $(\sigma - \sigma') > 0$
by (*meson Diff-eq-empty-iff card-0-eq finite-Diff gr0I state-is-finite*)
have $\forall\ n.\ \forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc n* $=$ *card* $(\sigma - \sigma') \longrightarrow\ \sigma \cup \sigma' \in \Sigma$
apply (*rule*)
proof −
fix *n*
show $\forall \sigma{\in}\Sigma.\ \forall \sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc n* $=$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
apply (*induction n*)
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ \sigma'$
assume $\sigma \in \Sigma$ **and** $\sigma' \in \Sigma$ **and** $\neg\ \sigma \subseteq \sigma' \wedge$ *Suc 0* $=$ *card* $(\sigma - \sigma')$
then have *is-singleton* $(\sigma - \sigma')$
by (*simp add*: *is-singleton-altdef*)
then have $\{$*the-elem* $(\sigma - \sigma')\} \cup \sigma' \in \Sigma$
using ‹$\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)$› ‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$›
by (*metis Un-commute* ‹$\neg\ \sigma \subseteq \sigma' \wedge$ *Suc 0* $=$ *card* $(\sigma - \sigma')$› *is-singleton-the-elem singletonD*)
then show $\sigma \cup \sigma' \in \Sigma$
by (*metis Un-Diff-cancel2* ‹*is-singleton* $(\sigma - \sigma')$› *is-singleton-the-elem*)

next
show $\bigwedge n.\ \forall \sigma{\in}\Sigma.\ \forall \sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc n* $=$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma \Longrightarrow \forall \sigma{\in}\Sigma.\ \forall \sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* (*Suc n*) $=$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
apply (*rule*, *rule*, *rule*)
proof −
fix $n\ \sigma\ \sigma'$
assume $\forall \sigma{\in}\Sigma.\ \forall \sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc n* $=$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$ **and** $\sigma \in \Sigma$ **and** $\sigma' \in \Sigma$ **and** $\neg\ \sigma \subseteq \sigma' \wedge$ *Suc* (*Suc n*) $=$ *card* $(\sigma - \sigma')$
have $\forall\ m \in \sigma - \sigma'.\ \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge$ *Suc n* $=$ *card* $(\sigma - (\sigma' \cup \{m\}))$
using ‹$\neg\ \sigma \subseteq \sigma' \wedge$ *Suc* (*Suc n*) $=$ *card* $(\sigma - \sigma')$›
by (*metis Diff-eq-empty-iff Diff-insert Un-insert-right* ‹$\sigma \in \Sigma$›

add-diff-cancel-left′ card-0-eq card-Suc-Diff1 finite-Diff nat.simps(3) plus-1-eq-Suc state-is-finite sup-bot.right-neutral)
            have $\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma$
              using ‹$\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)$› ‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$› ‹$\neg\ \sigma \subseteq \sigma' \wedge$ *Suc* (*Suc* n) = *card* $(\sigma - \sigma')$›
              by *blast*
            then have $\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma \wedge \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge$ *Suc* n = *card* $(\sigma - (\sigma' \cup \{m\}))$
              using ‹$\forall\ m \in \sigma - \sigma'.\ \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge$ *Suc* n = *card* $(\sigma - (\sigma' \cup \{m\}))$›
              by *simp*
            then show $\sigma \cup \sigma' \in \Sigma$
              using ‹$\forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* n = *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$›
                by (*smt Un-Diff-cancel Un-commute Un-insert-right* ‹$\sigma \in \Sigma$› *insert-absorb2 mk-disjoint-insert sup-bot.right-neutral*)
          qed
        qed
      qed
      then show *?thesis*
        by (*meson* ‹$\forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists m{\in}\sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)$› *card-Suc-Diff1 finite-Diff state-is-finite*)
    qed
    then show *?thesis*
      using *False* ‹$\sigma 1 \in \Sigma$› ‹$\sigma 2 \in \Sigma$› **by** *blast*
  qed
qed

lemma (**in** *Protocol*) *union-of-finite-set-of-states-is-state* :
  $\forall\ \sigma\text{-}set \subseteq \Sigma.$ *finite* σ-*set* $\longrightarrow \bigcup\ \sigma\text{-}set \in \Sigma$
  apply *auto*
proof −
  have $\forall\ n.\ \forall\ \sigma\text{-}set \subseteq \Sigma.\ n =$ *card* σ-*set* $\longrightarrow$ *finite* σ-*set* $\longrightarrow \bigcup\ \sigma\text{-}set \in \Sigma$
    apply (*rule*)
  proof −
    fix n
    show $\forall\ \sigma\text{-}set \subseteq \Sigma.\ n =$ *card* σ-*set* $\longrightarrow$ *finite* σ-*set* $\longrightarrow \bigcup\sigma\text{-}set \in \Sigma$
      apply (*induction n*)
      apply (*rule*, *rule*, *rule*, *rule*)
      apply (*simp add*: *empty-set-exists-in-*Σ)
      apply (*rule*, *rule*, *rule*, *rule*)
    proof −
      fix n σ-*set*
      assume $\forall\sigma\text{-}set{\subseteq}\Sigma.\ n =$ *card* σ-*set* $\longrightarrow$ *finite* σ-*set* $\longrightarrow \bigcup\sigma\text{-}set \in \Sigma$ **and** σ-*set* $\subseteq \Sigma$ **and** *Suc* n = *card* σ-*set* **and** *finite* σ-*set*
      then have $\forall\ \sigma \in \sigma\text{-}set.\ \sigma\text{-}set - \{\sigma\} \subseteq \Sigma \wedge \bigcup\ (\sigma\text{-}set - \{\sigma\}) \in \Sigma$
        using ‹σ-*set* $\subseteq \Sigma$› ‹*Suc* n = *card* σ-*set*› ‹$\forall\sigma\text{-}set{\subseteq}\Sigma.\ n =$ *card* σ-*set* $\longrightarrow$ *finite* σ-*set* $\longrightarrow \bigcup\sigma\text{-}set \in \Sigma$›
        by (*metis* (*mono-tags*, *lifting*) *Suc-inject card.remove finite-Diff insert-Diff*

insert-subset)
    then have $\forall\ \sigma \in \sigma\text{-}set.\ \sigma\text{-}set - \{\sigma\} \subseteq \Sigma \wedge \bigcup\ (\sigma\text{-}set - \{\sigma\}) \in \Sigma \wedge \bigcup\ (\sigma\text{-}set - \{\sigma\}) \cup \sigma \in \Sigma$
      using *union-of-two-states-is-state* ‹σ-*set* $\subseteq \Sigma$› **by** *auto*
    then show $\bigcup \sigma\text{-}set \in \Sigma$
      by (*metis Sup-bot-conv*(*1*) *Sup-insert Un-commute empty-set-exists-in-*Σ *insert-Diff*)
   qed
  qed
  then show $\bigwedge\sigma\text{-}set.\ \sigma\text{-}set \subseteq \Sigma \Longrightarrow$ *finite* σ-*set* $\Longrightarrow \bigcup \sigma\text{-}set \in \Sigma$
   by *blast*
qed

inductive (**in** *Protocol*) *MessagePath* :: *state* $\Rightarrow$ *state* $\Rightarrow$ *message list* $\Rightarrow$ *bool*
where
  P-nil: *MessagePath* σ σ []
| *P-cons*: $[\![$ *immediately-next-message* (σ, *m*); $\sigma \cup \{m\} \in \Sigma$; *MessagePath* ($\sigma \cup \{m\}$) σ' *list* $]\!] \Longrightarrow$ *MessagePath* σ σ' (*m* # *list*)

lemma (**in** *Protocol*) *exist-message-path-nonnil*:
  assumes $\sigma' \in$ *futures* σ
  and $\sigma \in \Sigma$ $\sigma' \in \Sigma$ $\sigma \neq \sigma'$
  obtains *message-list* **where** *MessagePath* σ σ' *message-list*
proof−
  have *finite* σ $\wedge$ *finite* σ'
   using *assms*(*1*)
   unfolding *futures-def* Σ*t-def*
   using *rev-finite-subset state-is-finite* **by** *fastforce*
  obtain *d* **where** $d = \sigma' - \sigma$
   by *auto*
  have *finite d*
   by (*simp add*: ‹$d = \sigma' - \sigma$› ‹*finite* σ $\wedge$ *finite* σ'›)

  assume $\bigwedge$*message-list*. *MessagePath* σ σ' *message-list* $\Longrightarrow$ *thesis*

  {
   fix *n*
   have $[\![$ $n =$ *card* ($\sigma' - \sigma$); *finite* σ; *finite* σ'; $\sigma \neq \sigma'$; $\sigma \in \Sigma$; $\sigma' \in \Sigma$; $\sigma \subseteq \sigma'$ $]\!]$ $\Longrightarrow \exists$*message-list*. *MessagePath* σ σ' *message-list* $\wedge$ *length message-list* $= n$
    apply (*induct n arbitrary*: σ σ' *d*)
    apply (*simp*)
   proof *simp*
    fix *n* **and** σ :: *state* **and** σ'
    assume $\bigwedge\sigma\ \sigma'$.

```
          n = card (σ' − σ) ⟹
          finite σ ⟹
          finite σ' ⟹
          σ ≠ σ' ⟹
           σ ∈ Σ ⟹ σ' ∈ Σ ⟹ σ ⊆ σ' ⟹ ∃ message-list. MessagePath σ σ'
message-list ∧ length message-list = card (σ' − σ)
        and Suc n = card (σ' − σ) finite σ finite σ' σ ≠ σ' σ ∈ Σ σ' ∈ Σ σ ⊆ σ'

      obtain m where immediately-next-message (σ, m) m ∈ σ' − σ
        using state-differences-have-immediately-next-messages
      by (meson ‹σ ≠ σ'› ‹σ ⊆ σ'› ‹σ' ∈ Σ› psubsetI strict-subset-of-state-have-immediately-next-messages)
      have cardn: card (σ' − (σ ∪ {m})) = n
        using ‹Suc n = card (σ' − σ)› ‹finite σ'› ‹m ∈ σ' − σ› by auto
      have finite (σ ∪ {m})
        by (simp add: ‹finite σ›)
      have σ ∪ {m} ∈ Σ
         using ‹σ ∈ Σ› ‹σ' ∈ Σ› ‹immediately-next-message (σ, m)› ‹m ∈ σ' − σ›
message-in-state-is-valid state-transition-by-immediately-next-message by fastforce
      have σ ∪ {m} ⊆ σ'
        using ‹σ ⊆ σ'› ‹m ∈ σ' − σ› by auto
      have m ∉ σ
        using ‹m ∈ σ' − σ› by auto

    obtain message-list where MessagePath σ σ' message-list ∧ length message-list
= Suc n
      proof (cases σ ∪ {m} = σ')
        case True
       assume ⋀message-list. MessagePath σ σ' message-list ∧ length message-list
= Suc n ⟹ thesis
        have σ' − (σ ∪ {m}) = ∅
          using True by auto
        hence card (σ' − (σ ∪ {m})) = 0
          by (metis card-empty)
        hence n = 0
          using cardn by simp
        have MessagePath σ σ' [m]
          using P-cons True ‹immediately-next-message (σ, m)›
          using P-nil ‹σ ∪ {m} ∈ Σ› by auto
        have length [m] = Suc 0
          by simp
        show ?thesis
          using ‹MessagePath σ σ' [m]› ‹length [m] = Suc 0› ‹n = 0› that by blast
      next
        case False
       assume ⋀message-list. MessagePath σ σ' message-list ∧ length message-list
= Suc n ⟹ thesis
       obtain prev-list where MessagePath (σ ∪ {m}) σ' prev-list ∧ length prev-list
= card (σ' − (σ ∪ {m}))
           using False ‹⋀σ' σ. ⟦n = card (σ' − σ); finite σ; finite σ'; σ ≠ σ'; σ
```

∈ Σ; σ′ ∈ Σ; σ ⊆ σ′⟧ ⟹ ∃ *message-list*. *MessagePath σ σ′ message-list* ∧ *length message-list* = *card* (σ′ − σ)› ‹σ ∪ {*m*} ∈ Σ› ‹σ ∪ {*m*} ⊆ σ′› ‹σ′ ∈ Σ› ‹*finite* (σ ∪ {*m*})› ‹*finite* σ′› *cardn* **by** *blast*

have *MessagePath σ σ′* (*m* # *prev-list*)
using *P-cons* ‹*MessagePath* (σ ∪ {*m*}) σ′ *prev-list* ∧ *length prev-list* = *card* (σ′ − (σ ∪ {*m*}))› ‹*immediately-next-message* (σ, *m*)›
‹σ ∪ {*m*} ∈ Σ› **by** *blast*
then show *?thesis*
using ‹*MessagePath* (σ ∪ {*m*}) σ′ *prev-list* ∧ *length prev-list* = *card* (σ′ − (σ ∪ {*m*}))› *cardn length-Suc-conv that* **by** *blast*
qed

then show ∃ *message-list*. *MessagePath σ σ′ message-list* ∧ *length message-list* = *card* (σ′ − σ)
using ‹*Suc n* = *card* (σ′ − σ)› **by** *auto*
qed
}

then show *?thesis*
using *Params.state-is-finite assms*(*1*) *assms*(*2*) *assms*(*3*) *assms*(*4*) *futures-def that* **by** *fastforce*
qed

lemma (**in** *Protocol*) *coherent-nonnil-message-path*:
assumes *MessagePath σ σ′ message-list length message-list* ≠ *0*
obtains *m ms* **where** *message-list* = *m* # *ms MessagePath* (σ ∪ {*m*}) σ′ *ms immediately-next-message* (σ,*m*) σ ∪ {*m*} ∈ Σ
using *assms*
apply (*cases rule*: *MessagePath.cases*)
apply *simp*
by *blast*

lemma (**in** *Protocol*) *coherent-nil-message-path*: *MessagePath σ σ′* [] ⟹ σ = σ′
using *MessagePath.cases* **by** *blast*

lemma (**in** *Protocol*) *coherent-message-path-inclusive*: *MessagePath σ σ′ ms* ⟹ σ ⊆ σ′
by (*induct rule*: *MessagePath.induct*, *auto*)

lemma (**in** *Protocol*) *exist-message-path*:
assumes σ′ ∈ *futures* σ σ ∈ Σ σ′ ∈ Σ
obtains *message-list* **where** *MessagePath σ σ′ message-list*
using *P-nil Protocol.exist-message-path-nonnil Protocol-axioms assms*(*1*) *assms*(*2*) *assms*(*3*) **by** *blast*

definition (**in** *Protocol*) *minimal-transitions* :: (*state* $*$ *state*) *set*
  where
    minimal-transitions $\equiv \{(\sigma, \sigma') \mid \sigma\ \sigma'.\ \sigma \in \Sigma t \wedge \sigma' \in \Sigma t \wedge$ *is-future-state* $(\sigma, \sigma') \wedge \sigma \neq \sigma'$
      $\wedge\ (\nexists\ \sigma''.\ \sigma'' \in \Sigma \wedge$ *is-future-state* $(\sigma, \sigma'') \wedge$ *is-future-state* $(\sigma'', \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma')\}$

lemma *non-empty-non-singleton-imps-two-elements* :
  $A \neq \emptyset \Longrightarrow \neg$ *is-singleton* $A \Longrightarrow \exists\ a1\ a2.\ a1 \neq a2 \wedge \{a1, a2\} \subseteq A$
  by (*metis inf.orderI inf-bot-left insert-subset is-singletonI'*)

lemma (**in** *Protocol*) *minimal-transition-implies-recieving-single-message* :
  $\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow$ *is-singleton* $(\sigma' - \sigma)$
proof (*rule ccontr*)
  assume $\neg$ ($\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow$ *is-singleton* $(\sigma' - \sigma)$)
  then have $\exists\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$
    by *blast*
  have $\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow$
      $(\nexists\ \sigma''.\ \sigma'' \in \Sigma \wedge$ *is-future-state* $(\sigma, \sigma'') \wedge$ *is-future-state* $(\sigma'', \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma')$
    by (*simp add*: *minimal-transitions-def*)
  have $\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$
    $\longrightarrow (\exists\ m1\ m2.\ \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma, m1))$
    apply (*rule*, *rule*, *rule*)
  proof −
    fix $\sigma\ \sigma'$
    assume $(\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$
    then have $\sigma' - \sigma \neq \emptyset$
      apply (*simp add*: *minimal-transitions-def*)
      by *blast*
    have $\sigma' \in \Sigma \wedge \sigma \in \Sigma \wedge$ *is-future-state* (σ, σ')
      using ‹$(\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$›
      by (*simp add*: *minimal-transitions-def* Σ*t-def*)
    then have $\sigma' - \sigma \subseteq M$
      using *state-difference-is-valid-message* **by** *auto*
    then have $\exists\ m1\ m2.\ \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2$
      using *non-empty-non-singleton-imps-two-elements*
        ‹$(\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$› ‹$\sigma' - \sigma \neq \emptyset$›
      by (*metis* (*full-types*) *contra-subsetD insert-subset subsetI*)
    then show $\exists\ m1\ m2.\ \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma, m1)$
      using *state-differences-have-immediately-next-messages*
      by (*metis Diff-iff* ‹$\sigma' \in \Sigma \wedge \sigma \in \Sigma \wedge$ *is-future-state* (σ, σ')› *insert-subset message-in-state-is-valid*)

qed
have $\forall\ \sigma\ \sigma'.\ (\sigma,\ \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma) \longrightarrow$
$(\exists\ \sigma''.\ \sigma'' \in \Sigma \wedge$ *is-future-state* $(\sigma,\ \sigma'') \wedge$ *is-future-state* $(\sigma'',\ \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma')$
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ \sigma'$
assume $(\sigma,\ \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$
then have $\exists\ m1\ m2.\ \{m1,\ m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma,\ m1)$
using ‹$\forall\ \sigma\ \sigma'.\ (\sigma,\ \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$ $\longrightarrow (\exists\ m1\ m2.\ \{m1,\ m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma,\ m1))$›
by *simp*
then obtain $m1\ m2$ **where** $\{m1,\ m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma,\ m1)$
by *auto*
have $\sigma \in \Sigma \wedge \sigma' \in \Sigma$
using ‹$(\sigma,\ \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$›
by (*simp add*: *minimal-transitions-def* Σ*t-def*)
then have $\sigma \cup \{m1\} \in \Sigma$
using ‹$\{m1,\ m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma,\ m1)$›
state-transition-by-immediately-next-message
by *simp*
have *is-future-state* $(\sigma,\ \sigma \cup \{m1\}) \wedge$ *is-future-state* $(\sigma \cup \{m1\},\ \sigma')$
using ‹$(\sigma,\ \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$› ‹$\{m1,\ m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma,\ m1)$› *minimal-transitions-def* **by** *auto*
have $\sigma \neq \sigma \cup \{m1\} \wedge \sigma \cup \{m1\} \neq \sigma'$
using ‹$\{m1,\ m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma,\ m1)$› **by** *auto*
then show $\exists \sigma''.\ \sigma'' \in \Sigma \wedge$ *is-future-state* $(\sigma,\ \sigma'') \wedge$ *is-future-state* $(\sigma'',\ \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma'$
using ‹$\sigma \cup \{m1\} \in \Sigma$› ‹*is-future-state* $(\sigma,\ \sigma \cup \{m1\}) \wedge$ *is-future-state* $(\sigma \cup \{m1\},\ \sigma')$›
by *auto*
qed
then show *False*
using ‹$\forall \sigma\ \sigma'.\ (\sigma,\ \sigma') \in$ *minimal-transitions* $\longrightarrow (\nexists \sigma''.\ \sigma'' \in \Sigma \wedge$ *is-future-state* $(\sigma,\ \sigma'') \wedge$ *is-future-state* $(\sigma'',\ \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma')$› ‹$\neg (\forall \sigma\ \sigma'.\ (\sigma,\ \sigma') \in$ *minimal-transitions* $\longrightarrow$ *is-singleton* $(\sigma' - \sigma))$› **by** *blast*
qed

lemma (**in** *Protocol*) *minimal-transitions-reconstruction* :
$\forall\ \sigma\ \sigma'.\ (\sigma,\ \sigma') \in$ *minimal-transitions* $\longrightarrow \sigma \cup \{$*the-elem* $(\sigma' - \sigma)\} = \sigma'$
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ \sigma'$

assume $(\sigma, \sigma') \in$ *minimal-transitions*
then have *is-singleton* $(\sigma' - \sigma)$
using *minimal-transitions-def minimal-transition-implies-recieving-single-message*
by *auto*
then have $\sigma \subseteq \sigma'$
using ‹$(\sigma, \sigma') \in$ *minimal-transitions*› *minimal-transitions-def* **by** *auto*
then show $\sigma \cup \{$*the-elem* $(\sigma' - \sigma)\} = \sigma'$
by (*metis Diff-partition* ‹*is-singleton* $(\sigma' - \sigma)$› *is-singleton-the-elem*)
qed

lemma (**in** *Protocol*) *minimal-transition-is-immediately-next-message* :
$\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longleftrightarrow$ *immediately-next-message* $(\sigma,$ *the-elem*
$(\sigma' - \sigma))$
proof −
have $\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow$ *immediately-next-message* $(\sigma,$
the-elem $(\sigma' - \sigma))$
using *minimal-transition-implies-recieving-single-message state-transition-only-made-by-immediately-next-m*
state-differences-have-immediately-next-messages
state-difference-is-valid-message
apply (*simp add*: *minimal-transitions-def immediately-next-message-def*)

oops

end

4 Safety Proof

theory *ConsensusSafety*

imports *Main CBCCasper MessageJustification StateTransition Libraries/LaTeXsugar*

begin

lemma (**in** *Protocol*) *monotonic-futures* :
$\forall\ \sigma'\ \sigma.\ \sigma' \in \Sigma t \wedge \sigma \in \Sigma t$
$\longrightarrow \sigma' \in$ *futures* $\sigma \longleftrightarrow$ *futures* $\sigma' \subseteq$ *futures* σ
apply (*simp add*: *futures-def*) **by** *auto*

theorem (**in** *Protocol*) *two-party-common-futures* :
$\forall\ \sigma 1\ \sigma 2.\ \sigma 1 \in \Sigma t \wedge \sigma 2 \in \Sigma t$
$\longrightarrow$ *is-faults-lt-threshold* $(\sigma 1 \cup \sigma 2)$
$\longrightarrow$ *futures* $\sigma 1 \cap$ *futures* $\sigma 2 \neq \emptyset$
apply (*simp add*: *futures-def* Σt*-def*) **using** *union-of-two-states-is-state*

by *blast*

theorem (**in** *Protocol*) *n-party-common-futures* :
  $\forall$ *σ-set*. *σ-set* $\subseteq \Sigma t$
  $\longrightarrow$ *finite σ-set*
  $\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
  $\longrightarrow \bigcap$ {*futures σ* | *σ*. *σ* $\in$ *σ-set*} $\neq \emptyset$
  apply (*simp add*: *futures-def Σt-def*) **using** *union-of-finite-set-of-states-is-state*
  by *blast*

lemma (**in** *Protocol*) *n-party-common-futures-exists* :
  $\forall$ *σ-set*. *σ-set* $\subseteq \Sigma t$
  $\longrightarrow$ *finite σ-set*
  $\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
  $\longrightarrow$ ($\exists$ *σ* $\in \Sigma t$. *σ* $\in \bigcap$ {*futures σ* | *σ*. *σ* $\in$ *σ-set*})
  apply (*simp add*: *futures-def Σt-def*) **using** *union-of-finite-set-of-states-is-state*
  by *blast*

definition (**in** *Protocol*) *state-property-is-decided* :: (*state-property* $*$ *state*) $\Rightarrow$ *bool*
  where
    state-property-is-decided = (λ(*p*, *σ*). ($\forall$ *σ'* $\in$ *futures σ* . *p σ'*))

lemma (**in** *Protocol*) *forward-consistency* :
  $\forall$ *σ' σ*. *σ'* $\in \Sigma t \wedge$ *σ* $\in \Sigma t$
  $\longrightarrow$ *σ'* $\in$ *futures σ*
  $\longrightarrow$ *state-property-is-decided* (*p*, *σ*)
  $\longrightarrow$ *state-property-is-decided* (*p*, *σ'*)
  apply (*simp add*: *futures-def state-property-is-decided-def*)
  by *auto*

fun *state-property-not* :: *state-property* $\Rightarrow$ *state-property*
  where
    state-property-not p = (λ*σ*. ($\neg$ *p σ*))

lemma (**in** *Protocol*) *backword-consistency* :
  $\forall$ *σ' σ*. *σ'* $\in \Sigma t \wedge$ *σ* $\in \Sigma t$
  $\longrightarrow$ *σ'* $\in$ *futures σ*
  $\longrightarrow$ *state-property-is-decided* (*p*, *σ'*)
  $\longrightarrow \neg$*state-property-is-decided* (*state-property-not p*, *σ*)
  apply (*simp add*: *futures-def state-property-is-decided-def*)
  by *auto*

theorem (**in** *Protocol*) *two-party-consensus-safety-for-state-property* :
  $\forall$ *σ1 σ2*. *σ1* $\in$ *Σt* $\wedge$ *σ2* $\in$ *Σt*
  $\longrightarrow$ *is-faults-lt-threshold* (*σ1* $\cup$ *σ2*)
  $\longrightarrow$ $\neg$(*state-property-is-decided* (*p*, *σ1*) $\wedge$ *state-property-is-decided* (*state-property-not p*, *σ2*))
  apply (*simp add*: *state-property-is-decided-def*)
  using *two-party-common-futures*
  by (*metis Int-emptyI*)

definition (**in** *Protocol*) *state-properties-are-inconsistent* :: *state-property set* $\Rightarrow$ *bool*
  where
    state-properties-are-inconsistent p-set = ($\forall$ *σ* $\in$ *Σ*. $\neg$ ($\forall$ *p* $\in$ *p-set*. *p σ*))

definition (**in** *Protocol*) *state-properties-are-consistent* :: *state-property set* $\Rightarrow$ *bool*
  where
    state-properties-are-consistent p-set = ($\exists$ *σ* $\in$ *Σ*. $\forall$ *p* $\in$ *p-set*. *p σ*)

definition (**in** *Protocol*) *state-property-decisions* :: *state* $\Rightarrow$ *state-property set*
  where
    state-property-decisions σ = {*p*. *state-property-is-decided* (*p*, *σ*)}

theorem (**in** *Protocol*) *n-party-safety-for-state-properties* :
  $\forall$ *σ-set*. *σ-set* $\subseteq$ *Σt*
  $\longrightarrow$ *finite σ-set*
  $\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
  $\longrightarrow$ *state-properties-are-consistent* ($\bigcup$ {*state-property-decisions σ* | *σ*. *σ* $\in$ *σ-set*})
  apply *rule+*
proof−
  fix *σ-set*
  assume *σ-set*: *σ-set* $\subseteq$ *Σt*
  and *finite σ-set*
  and *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
  hence $\exists$*σ*$\in$*Σt*. *σ* $\in$ $\bigcap$ {*futures σ* | *σ*. *σ* $\in$ *σ-set*}
    using *n-party-common-futures-exists* **by** *simp*
  hence $\exists$*σ*$\in$*Σt*. $\forall$*s*$\in$*σ-set*. *σ* $\in$ *futures s*
    by *blast*
  hence $\exists$*σ*$\in$*Σt*. ($\forall$*s*$\in$*σ-set*. *σ* $\in$ *futures s*) $\wedge$ ($\forall$*s*$\in$*σ-set*. *σ* $\in$ *futures s* $\longrightarrow$ ($\forall$*p*. *state-property-is-decided* (*p*,*s*) $\longrightarrow$ *state-property-is-decided* (*p*,*σ*)))
    by (*simp add*: *subset-eq state-property-is-decided-def futures-def*)
  hence $\exists$*σ*$\in$*Σt*. $\forall$*s*$\in$*σ-set*. ($\forall$*p*. *state-property-is-decided* (*p*,*s*) $\longrightarrow$ *state-property-is-decided* (*p*,*σ*))
    by *blast*
  hence $\exists$*σ*$\in$*Σt*. $\forall$*s*$\in$*σ-set*. ($\forall$*p* $\in$ *state-property-decisions s*. *state-property-is-decided*

$(p,\sigma))$
  by (*simp add*: *state-property-decisions-def*)
 hence $\exists\,\sigma\in\Sigma t.\ \forall\,p\in\bigcup\{$*state-property-decisions* $\sigma\mid\sigma.\ \sigma\in\sigma$*-set*$\}$. *state-property-is-decided* (p,σ)
 proof−
  obtain σ **where** $\sigma\in\Sigma t\ \forall\,s\in\sigma$*-set*. $(\forall\,p\in$ *state-property-decisions* s. *state-property-is-decided* $(p,\sigma))$
   using ‹$\exists\,\sigma\in\Sigma t.\ \forall\,s\in\sigma$*-set*. $\forall\,p\in$*state-property-decisions* s. *state-property-is-decided* $(p,\ \sigma)$› **by** *blast*
  have $\forall\,p\in\bigcup\{$*state-property-decisions* $\sigma\mid\sigma.\ \sigma\in\sigma$*-set*$\}$. *state-property-is-decided* (p,σ)
   using ‹$\forall\,s\in\sigma$*-set*. $\forall\,p\in$*state-property-decisions* s. *state-property-is-decided* $(p,\ \sigma)$› **by** *fastforce*
  thus *?thesis*
   using ‹$\sigma\in\Sigma t$› **by** *blast*
 qed
 hence $\exists\,\sigma\in\Sigma t.\ \forall\,p\in\bigcup\{$*state-property-decisions* $\sigma\mid\sigma.\ \sigma\in\sigma$*-set*$\}$. $\forall\,\sigma'\in$*futures* $\sigma.\ p\ \sigma'$
  by (*simp add*: *state-property-decisions-def futures-def state-property-is-decided-def*)
 show *state-properties-are-consistent* $(\bigcup\{$*state-property-decisions* $\sigma\mid\sigma.\ \sigma\in\sigma$*-set*$\})$
  unfolding *state-properties-are-consistent-def*
  by (*metis* (*mono-tags*, *lifting*) Σt*-def* ‹$\exists\,\sigma\in\Sigma t.\ \forall\,p\in\bigcup\{$*state-property-decisions* $\sigma\mid\sigma.\ \sigma\in\sigma$*-set*$\}$. $\forall\,\sigma'\in$*futures* $\sigma.\ p\ \sigma'$› *mem-Collect-eq monotonic-futures order-refl*)
qed

definition (**in** *Protocol*) *naturally-corresponding-state-property* :: *consensus-value-property* $\Rightarrow$ *state-property*
 where
  naturally-corresponding-state-property $q = (\lambda\sigma.\ \forall\ c\in\varepsilon\ \sigma.\ q\ c)$

definition (**in** *Protocol*) *consensus-value-properties-are-consistent* :: *consensus-value-property set* $\Rightarrow$ *bool*
 where
  consensus-value-properties-are-consistent *q-set* $= (\exists\ c\in C.\ \forall\ q\in$ *q-set*. $q\ c)$

lemma (**in** *Protocol*) *naturally-corresponding-consistency* :
 $\forall$ *q-set*. *state-properties-are-consistent* $\{$*naturally-corresponding-state-property* $q\mid q.\ q\in$ *q-set*$\}$
 $\longrightarrow$ *consensus-value-properties-are-consistent* *q-set*
 apply (*rule*, *rule*)
proof −
 fix *q-set*
 have
  state-properties-are-consistent $\{$*naturally-corresponding-state-property* $q\mid q.\ q$

```
∈ q-set}
    ⟶ (∃ σ ∈ Σ. ∀ p ∈ {λσ'. ∀ c ∈ ε σ'. q c | q. q ∈ q-set}. p σ)
   by (simp add: naturally-corresponding-state-property-def state-properties-are-consistent-def)
  moreover have
    (∃ σ ∈ Σ. ∀ p ∈ {λσ'. ∀ c ∈ ε σ'. q c | q. q ∈ q-set}. p σ)
    ⟶ (∃ σ ∈ Σ. ∀ q' ∈ q-set. (λσ'. ∀ c ∈ ε σ'. q' c) σ)
    by (metis (mono-tags, lifting) mem-Collect-eq)
  moreover have
    (∃ σ ∈ Σ. ∀ q ∈ q-set. (λσ'. ∀ c ∈ ε σ'. q c) σ)
    ⟶ (∃ σ ∈ Σ. ∀ q' ∈ q-set. ∀ c ∈ ε σ. q' c)
    by blast
  moreover have
    (∃ σ ∈ Σ. ∀ q ∈ q-set. ∀ c ∈ ε σ. q c)
    ⟶ (∃ σ ∈ Σ. ∀ c ∈ ε σ. ∀ q' ∈ q-set. q' c)
    by blast
  moreover have
    (∃ σ ∈ Σ. ∀ c ∈ ε σ. ∀ q ∈ q-set. q c)
    ⟶ (∃ σ ∈ Σ. ∃ c ∈ ε σ. ∀ q' ∈ q-set. q' c)
    by (meson all-not-in-conv estimates-are-non-empty)
  moreover have
    (∃ σ ∈ Σ. ∃ c ∈ ε σ. ∀ q ∈ q-set. q c)
    ⟶ (∃ c ∈ C. ∀ q' ∈ q-set. q' c)
    using is-valid-estimator-def ε-type by fastforce
  ultimately show
    state-properties-are-consistent {naturally-corresponding-state-property q |q. q ∈
q-set}
    ⟹ consensus-value-properties-are-consistent q-set
    by (simp add: consensus-value-properties-are-consistent-def)
qed


definition (in Protocol) consensus-value-property-is-decided :: (consensus-value-property
∗ state) ⇒ bool
  where
    consensus-value-property-is-decided
      = (λ(q, σ). state-property-is-decided (naturally-corresponding-state-property q,
σ))


definition (in Protocol) consensus-value-property-decisions :: state ⇒ consensus-value-property
set
  where
    consensus-value-property-decisions σ = {q. consensus-value-property-is-decided
(q, σ)}


theorem (in Protocol) n-party-safety-for-consensus-value-properties :
  ∀ σ-set. σ-set ⊆ Σt
  ⟶ finite σ-set
```

$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
$\longrightarrow$ *consensus-value-properties-are-consistent* ($\bigcup$ {*consensus-value-property-decisions* *σ* | *σ*. *σ* ∈ *σ-set*})
apply (*rule*, *rule*, *rule*, *rule*)
proof −
fix *σ-set*
assume *σ-set* ⊆ Σ*t*
and *finite σ-set*
and *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
hence *state-properties-are-consistent* ($\bigcup$ {*state-property-decisions σ* | *σ*. *σ* ∈ *σ-set*})
using ‹*σ-set* ⊆ Σ*t*› *n-party-safety-for-state-properties* **by** *auto*
hence *state-properties-are-consistent* {*p* ∈ $\bigcup$ {*state-property-decisions σ* | *σ*. *σ* ∈ *σ-set*}. ∃ *q*. *p* = *naturally-corresponding-state-property q*}
unfolding *naturally-corresponding-state-property-def state-properties-are-consistent-def*
apply (*simp*)
by *meson*
hence *state-properties-are-consistent* {*naturally-corresponding-state-property q* | *q*. *naturally-corresponding-state-property q* ∈ $\bigcup$ {*state-property-decisions σ* | *σ*. *σ* ∈ *σ-set*}}
by (*smt Collect-cong*)
hence *consensus-value-properties-are-consistent* {*q*. *naturally-corresponding-state-property q* ∈ $\bigcup$ {*state-property-decisions σ* | *σ*. *σ* ∈ *σ-set*}}
using *naturally-corresponding-consistency*
proof −
show *?thesis*
by (*metis* (*no-types*) *Setcompr-eq-image* ‹∀ *q-set*. *state-properties-are-consistent* {*naturally-corresponding-state-property q* |*q*. *q* ∈ *q-set*} $\longrightarrow$ *consensus-value-properties-are-consistent q-set*› ‹*state-properties-are-consistent* {*naturally-corresponding-state-property q* |*q*. *naturally-corresponding-state-property q* ∈ $\bigcup${*state-property-decisions σ* |*σ*. *σ* ∈ *σ-set*}}› *setcompr-eq-image*)
qed
hence *consensus-value-properties-are-consistent* ($\bigcup$ {*consensus-value-property-decisions* *σ* | *σ*. *σ* ∈ *σ-set*})
apply (*simp add*: *consensus-value-property-decisions-def consensus-value-property-is-decided-def state-property-decisions-def consensus-value-properties-are-consistent-def*)
by (*metis mem-Collect-eq*)
thus
consensus-value-properties-are-consistent ($\bigcup$ {*consensus-value-property-decisions* *σ* | *σ*. *σ* ∈ *σ-set*})
by *simp*
qed

fun *consensus-value-property-not* :: *consensus-value-property* ⇒ *consensus-value-property*
where
consensus-value-property-not p = (λ*c*. (¬ *p c*))

lemma (**in** *Protocol*) *negation-is-not-decided-by-other-validator* :
∀ *σ-set*. *σ-set* ⊆ Σ*t*

$\longrightarrow$ *finite σ-set*
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
$\longrightarrow$ (∀ *σ σ′ p*. {*σ*, *σ′*} ⊆ *σ-set* ∧ *p* ∈ *consensus-value-property-decisions σ*
$\longrightarrow$ *consensus-value-property-not p* ∉ *consensus-value-property-decisions*
σ′)
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof –
fix *σ-set σ σ′ p*
assume *σ-set* ⊆ Σ*t* **and** *finite σ-set* **and** *is-faults-lt-threshold* ($\bigcup$*σ-set*) **and** {*σ*, *σ′*} ⊆ *σ-set* ∧ *p* ∈ *consensus-value-property-decisions σ*
hence ∃ *σ*. *σ* ∈ Σ*t* ∧ *σ* ∈ $\bigcap$ {*futures σ* | *σ*. *σ* ∈ *σ-set*}
using *n-party-common-futures-exists* **by** *meson*
then obtain *σ″* **where** *σ″* ∈ Σ*t* ∧ *σ″* ∈ $\bigcap$ {*futures σ* | *σ*. *σ* ∈ *σ-set*} **by** *auto*
hence *state-property-is-decided* (*naturally-corresponding-state-property p*, *σ″*)
using ‹{*σ*, *σ′*} ⊆ *σ-set* ∧ *p* ∈ *consensus-value-property-decisions σ*› *consensus-value-property-decisions-def consensus-value-property-is-decided-def*
using ‹*σ-set* ⊆ Σ*t*› *forward-consistency* **by** *fastforce*
have *σ″* ∈ *futures σ′*
using ‹*σ″* ∈ Σ*t* ∧ *σ″* ∈ $\bigcap$ {*futures σ* | *σ*. *σ* ∈ *σ-set*}› ‹{*σ*, *σ′*} ⊆ *σ-set* ∧ *p* ∈ *consensus-value-property-decisions σ*›
by *auto*
hence ¬ *state-property-is-decided* (*state-property-not* (*naturally-corresponding-state-property* *p*), *σ′*)

using *backword-consistency* ‹*state-property-is-decided* (*naturally-corresponding-state-property* *p*, *σ″*)›
using ‹*σ″* ∈ Σ*t* ∧ *σ″* ∈ $\bigcap$ {*futures σ* | *σ*. *σ* ∈ *σ-set*}› ‹*σ-set* ⊆ Σ*t*› ‹{*σ*, *σ′*} ⊆ *σ-set* ∧ *p* ∈ *consensus-value-property-decisions σ*› **by** *auto*
then show *consensus-value-property-not p* ∉ *consensus-value-property-decisions* *σ′*
apply (*simp add*: *consensus-value-property-decisions-def consensus-value-property-is-decided-def naturally-corresponding-state-property-def state-property-is-decided-def*)
using Σ*t-def estimates-are-non-empty futures-def* **by** *fastforce*
qed

lemma (**in** *Protocol*) *n-party-consensus-safety* :
∀ *σ-set*. *σ-set* ⊆ Σ*t*
$\longrightarrow$ *finite σ-set*
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
$\longrightarrow$ (∀ *p* ∈ $\bigcup$ {*consensus-value-property-decisions σ′* | *σ′*. *σ′* ∈ *σ-set*}.
(*λc*. (¬ *p c*)) ∉ $\bigcup$ {*consensus-value-property-decisions σ′* | *σ′*. *σ′* ∈ *σ-set*})
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof –
fix *σ-set p*
assume *σ-set* ⊆ Σ*t* **and** *finite σ-set* **and** *is-faults-lt-threshold* ($\bigcup$*σ-set*) **and** *p* ∈ $\bigcup$ {*consensus-value-property-decisions σ′* | *σ′*. *σ′* ∈ *σ-set*}
and (*λc*. (¬ *p c*)) ∈ $\bigcup$ {*consensus-value-property-decisions σ′* | *σ′*. *σ′* ∈ *σ-set*}
hence ∃ *σ*. *σ* ∈ Σ*t* ∧ *σ* ∈ $\bigcap$ {*futures σ* | *σ*. *σ* ∈ *σ-set*}

using *n-party-common-futures-exists* **by** *meson*
then obtain σ'' **where** $\sigma'' \in \Sigma t \wedge \sigma'' \in \bigcap$ {*futures* σ | σ. $\sigma \in$ *σ-set*} **by** *auto*
hence *state-property-is-decided* (*naturally-corresponding-state-property* *p*, σ'')
using ‹*p* $\in \bigcup$ {*consensus-value-property-decisions* σ' | σ'. $\sigma' \in$ *σ-set*}› *consensus-value-property-decisions-de*
consensus-value-property-is-decided-def
using ‹*σ-set* $\subseteq \Sigma t$› *forward-consistency* **by** *fastforce*
have *state-property-is-decided* (*naturally-corresponding-state-property* ($\lambda c.$ ($\neg$ *p*
c)), σ'')
using ‹($\lambda c.$ ($\neg$ *p* *c*)) $\in \bigcup$ {*consensus-value-property-decisions* σ' | σ'. $\sigma' \in$
σ-set}› *consensus-value-property-decisions-def consensus-value-property-is-decided-def*

using ‹*σ-set* $\subseteq \Sigma t$› *forward-consistency* ‹$\sigma'' \in \Sigma t \wedge \sigma'' \in \bigcap$ {*futures* σ | σ. σ
$\in$ *σ-set*}› **by** *fastforce*
then show *False*
using ‹*state-property-is-decided* (*naturally-corresponding-state-property* *p*, σ'')›
apply (*simp add*: *state-property-is-decided-def naturally-corresponding-state-property-def*)
by (*meson* *Σt-is-subset-of-Σ* ‹$\sigma'' \in \Sigma t \wedge \sigma'' \in \bigcap$ {*futures* σ | σ. $\sigma \in$ *σ-set*}›
estimates-are-non-empty monotonic-futures order-refl subsetCE)
qed

lemma (**in** *Protocol*) *two-party-consensus-safety-for-consensus-value-property* :
$\forall$ *σ1 σ2*. *σ1* $\in \Sigma t \wedge$ *σ2* $\in \Sigma t$
$\longrightarrow$ *is-faults-lt-threshold* (*σ1* $\cup$ *σ2*)
$\longrightarrow$ *consensus-value-property-is-decided* (*p*, *σ1*)
$\longrightarrow \neg$ *consensus-value-property-is-decided* (*consensus-value-property-not* *p*, *σ2*)
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix *σ1 σ2*
have *two-party*: $\forall$ *σ1 σ2*. {*σ1*, *σ2*} $\subseteq \Sigma t$
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ {*σ1*, *σ2*})
$\longrightarrow$ *p* $\in$ *consensus-value-property-decisions* *σ1*
$\longrightarrow$ *consensus-value-property-not* *p* $\notin$ *consensus-value-property-decisions*
σ2
using *negation-is-not-decided-by-other-validator*
by (*meson finite.emptyI finite.insertI order-refl*)
assume *σ1* $\in \Sigma t \wedge$ *σ2* $\in \Sigma t$ **and** *is-faults-lt-threshold* (*σ1* $\cup$ *σ2*) **and** *consensus-value-property-is-decided*
(*p*, *σ1*)
then show $\neg$ *consensus-value-property-is-decided* (*consensus-value-property-not*
p, *σ2*)
using *two-party*
apply (*simp add*: *consensus-value-property-decisions-def*)
by *blast*
qed

lemma (**in** *Protocol*) *n-party-consensus-safety-for-power-set-of-decisions* :
$\forall$ *σ-set*. *σ-set* $\subseteq \Sigma t$
$\longrightarrow$ *finite* *σ-set*
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ *σ-set*)

$\longrightarrow$ ($\forall$ σ *p-set*. $\sigma \in \sigma$*-set* $\wedge$ *p-set* $\in$ *Pow* ($\bigcup$ {*consensus-value-property-decisions* σ' | σ'. $\sigma' \in \sigma$*-set*}) − {$\emptyset$}
$\longrightarrow$ (λc. $\neg$ ($\forall$ *p* $\in$ *p-set*. *p c*)) $\notin$ *consensus-value-property-decisions* σ)
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix σ*-set* σ *p-set*
assume σ*-set* $\subseteq \Sigma t$ **and** *finite* σ*-set* **and** *is-faults-lt-threshold* ($\bigcup\sigma$*-set*)
and $\sigma \in \sigma$*-set* $\wedge$ *p-set* $\in$ *Pow* ($\bigcup$ {*consensus-value-property-decisions* σ' | σ'. σ' $\in \sigma$*-set*}) − {$\emptyset$}
and (λc. $\neg$ ($\forall$ *p* $\in$ *p-set*. *p c*)) $\in$ *consensus-value-property-decisions* σ
hence $\exists$ σ. $\sigma \in \Sigma t$ $\wedge$ $\sigma \in \bigcap$ {*futures* σ | σ. $\sigma \in \sigma$*-set*}
using *n-party-common-futures-exists* **by** *meson*
then obtain σ' **where** $\sigma' \in \Sigma t$ $\wedge$ $\sigma' \in \bigcap$ {*futures* σ | σ. $\sigma \in \sigma$*-set*} **by** *auto*
hence $\forall$ *p* $\in$ *p-set*. $\exists$ $\sigma'' \in \sigma$*-set*. *state-property-is-decided* (*naturally-corresponding-state-property* *p*, σ'')
using ‹$\sigma \in \sigma$*-set* $\wedge$ *p-set* $\in$ *Pow* ($\bigcup$ {*consensus-value-property-decisions* σ' | σ'. $\sigma' \in \sigma$*-set*}) − {$\emptyset$}›
apply (*simp add*: *consensus-value-property-decisions-def consensus-value-property-is-decided-def*)
by *blast*
have $\forall$ $\sigma'' \in \sigma$*-set*. $\sigma' \in$ *futures* σ''
using ‹$\sigma' \in \Sigma t$ $\wedge$ $\sigma' \in \bigcap$ {*futures* σ | σ. $\sigma \in \sigma$*-set*}› **by** *blast*
hence $\forall$ *p* $\in$ *p-set*. *state-property-is-decided* (*naturally-corresponding-state-property* *p*, σ')
using *forward-consistency* ‹$\forall$ *p* $\in$ *p-set*. $\exists$ $\sigma'' \in \sigma$*-set*. *state-property-is-decided* (*naturally-corresponding-state-property p*, σ'')›
by (*meson* ‹$\sigma' \in \Sigma t \wedge \sigma' \in \bigcap$ {*futures* σ | σ. $\sigma \in \sigma$*-set*}› ‹σ*-set* $\subseteq \Sigma t$› *subsetCE*)
hence *state-property-is-decided* (*naturally-corresponding-state-property* (λc. $\forall$ *p* $\in$ *p-set*. *p c*), σ')
apply (*simp add*: *naturally-corresponding-state-property-def state-property-is-decided-def*)
by *auto*
then show *False*
using ‹(λc. $\neg$ ($\forall$ *p* $\in$ *p-set*. *p c*)) $\in$ *consensus-value-property-decisions* σ›
apply (*simp add*: *consensus-value-property-decisions-def consensus-value-property-is-decided-def naturally-corresponding-state-property-def state-property-is-decided-def*)
using Σ*t-is-subset-of-*Σ ‹$\sigma \in \sigma$*-set* $\wedge$ *p-set* $\in$ *Pow* ($\bigcup$ {*consensus-value-property-decisions* σ' | σ'. $\sigma' \in \sigma$*-set*}) − {$\emptyset$}› ‹$\sigma' \in \Sigma t$ $\wedge$ $\sigma' \in \bigcap$ {*futures* σ | σ. $\sigma \in \sigma$*-set*}› *estimates-are-non-empty monotonic-futures* **by** *fastforce*
qed

end
theory *CliqueOracle*

imports *Main CBCCasper LatestMessage StateTransition ConsensusSafety*

begin

definition *agreeing* :: (*consensus-value-property* $*$ *state* $*$ *validator*) $\Rightarrow$ *bool*
  where
    agreeing = (λ(*p*, σ, *v*). $\forall$ *c* $\in$ *L-H-E* σ *v*. *p c*)

definition *agreeing-validators* :: (*consensus-value-property* $*$ *state*) $\Rightarrow$ *validator set*
  where
    agreeing-validators = (λ(*p*, σ).{*v* $\in$ *observed-non-equivocating-validators* σ. *agreeing* (*p*, σ, *v*)})

lemma (**in** *Protocol*) *agreeing-validators-type* :
  $\forall$ $\sigma \in \Sigma$. *agreeing-validators* (*p*, σ) $\subseteq$ *V*
  apply (*simp add*: *observed-non-equivocating-validators-def agreeing-validators-def*)
  using *observed-type-for-state* **by** *auto*

lemma (**in** *Protocol*) *agreeing-validators-finite* :
  $\forall$ $\sigma \in \Sigma$. *finite* (*agreeing-validators* (*p*, σ))
  by (*meson V-type agreeing-validators-type rev-finite-subset*)

lemma (**in** *Protocol*) *agreeing-validators-are-observed-non-equivocating-validators*
:
  $\forall$ $\sigma \in \Sigma$. *agreeing-validators* (*p*, σ) $\subseteq$ *observed-non-equivocating-validators* σ
  by (*simp add*: *agreeing-validators-def*)

lemma (**in** *Protocol*) *agreeing-validators-are-not-equivocating* :
  $\forall$ $\sigma \in \Sigma$. *agreeing-validators* (*p*, σ) $\cap$ *equivocating-validators* $\sigma = \emptyset$
  using *agreeing-validators-are-observed-non-equivocating-validators*
      observed-non-equivocating-validators-are-not-equivocating
  by *blast*

definition (**in** *Params*) *disagreeing-validators* :: (*consensus-value-property* $*$ *state*)
$\Rightarrow$ *validator set*
  where
    disagreeing-validators = (λ(*p*, σ). *V* $-$ *agreeing-validators* (*p*, σ) $-$ *equivocating-validators*

σ)

lemma (**in** *Protocol*) *disagreeing-validators-type* :
 $\forall\ \sigma \in \Sigma.$ *disagreeing-validators* (p, σ) $\subseteq$ V
 apply (*simp add*: *disagreeing-validators-def*)
 by *auto*

lemma (**in** *Protocol*) *disagreeing-validators-are-non-observed-or-not-agreeing* :
 $\forall\ \sigma \in \Sigma.$ *disagreeing-validators* (p, σ) = $\{v \in V -$ *equivocating-validators* $\sigma.\ v \notin$ *observed* $\sigma \vee (\exists\ c \in$ *L-H-E* $\sigma\ v.\ \neg\ p\ c)\}$
 apply (*simp add*: *disagreeing-validators-def agreeing-validators-def observed-non-equivocating-validators-def agreeing-def*)
 by *blast*

lemma (**in** *Protocol*) *disagreeing-validators-include-not-agreeing-validators* :
 $\forall\ \sigma \in \Sigma.$ $\{v \in V -$ *equivocating-validators* $\sigma.\ \exists\ c \in$ *L-H-E* $\sigma\ v.\ \neg\ p\ c\} \subseteq$ *disagreeing-validators* (p, σ)
 using *disagreeing-validators-are-non-observed-or-not-agreeing* **by** *blast*

lemma (**in** *Protocol*) *weight-measure-agreeing-plus-equivocating* :
 $\forall\ \sigma \in \Sigma.$ *weight-measure* (*agreeing-validators* (p, σ) $\cup$ *equivocating-validators* σ) = *weight-measure* (*agreeing-validators* (p, σ)) + *equivocation-fault-weight* σ
 unfolding *equivocation-fault-weight-def*
 using *agreeing-validators-are-not-equivocating weight-measure-disjoint-plus agreeing-validators-finite equivocating-validators-is-finite*
 by *simp*

lemma (**in** *Protocol*) *disagreeing-validators-weight-combined* :
 $\forall\ \sigma \in \Sigma.$ *weight-measure* (*disagreeing-validators* (p, σ)) = *weight-measure* V − *weight-measure* (*agreeing-validators* (p, σ)) − *equivocation-fault-weight* σ
 unfolding *disagreeing-validators-def*
 using *weight-measure-agreeing-plus-equivocating*
 unfolding *equivocation-fault-weight-def*
 using *agreeing-validators-are-not-equivocating weight-measure-subset-minus agreeing-validators-finite equivocating-validators-is-finite*
 by (*smt Diff-empty Diff-iff Int-iff V-type agreeing-validators-type equivocating-validators-type finite-Diff old.prod.case subset-iff*)

lemma (**in** *Protocol*) *agreeing-validators-weight-combined* :
 $\forall\ \sigma \in \Sigma.$ *weight-measure* (*agreeing-validators* (p, σ)) = *weight-measure* V − *weight-measure* (*disagreeing-validators* (p, σ)) − *equivocation-fault-weight* σ
 using *disagreeing-validators-weight-combined*
 by *simp*

definition (**in** *Params*) *majority* :: (*validator set* ∗ *state*) ⇒ *bool*
 where
 majority = (λ(*v-set*, σ). (*weight-measure v-set* > (*weight-measure* (V − *equivocating-validators* σ)) *div 2*))

definition (**in** *Protocol*) *majority-driven* :: *consensus-value-property* $\Rightarrow$ *bool*
  where
    majority-driven p = ($\forall$ $\sigma \in \Sigma$. *majority* (*agreeing-validators* (p, σ), σ) $\longrightarrow$ ($\forall$ $c \in \varepsilon\ \sigma$. $p\ c$))

definition (**in** *Protocol*) *max-driven* :: *consensus-value-property* $\Rightarrow$ *bool*
  where
    max-driven p =
      ($\forall$ $\sigma \in \Sigma$. *weight-measure* (*agreeing-validators* (p, σ)) > *weight-measure* (*disagreeing-validators* (p, σ)) $\longrightarrow$ ($\forall$ $c \in \varepsilon\ \sigma$. $p\ c$))

definition (**in** *Protocol*) *max-driven-for-future* :: *consensus-value-property* $\Rightarrow$ *state* $\Rightarrow$ *bool*
  where
    max-driven-for-future p σ =
      ($\forall$ $\sigma' \in \Sigma$. *is-future-state* (σ, σ')
      $\longrightarrow$ *weight-measure* (*agreeing-validators* (p, σ')) > *weight-measure* (*disagreeing-validators* (p, σ')) $\longrightarrow$ ($\forall$ $c \in \varepsilon\ \sigma'$. $p\ c$))

definition *later-disagreeing-messages* :: (*consensus-value-property* $*$ *message* $*$ *validator* $*$ *state*) $\Rightarrow$ *message set*
  where
    later-disagreeing-messages = (λ(p, m, v, σ).{$m' \in$ *later-from* (m, v, σ). $\neg$ p (*est* m')})

lemma (**in** *Protocol*) *later-disagreeing-messages-type* :
  $\forall$ $p\ \sigma\ v\ m$. $\sigma \in \Sigma \land v \in V \land m \in M \longrightarrow$ *later-disagreeing-messages* (p, m, v, σ) $\subseteq M$
  unfolding *later-disagreeing-messages-def*
  using *later-from-type-for-state* **by** *auto*

definition *is-clique* :: (*validator set* $*$ *consensus-value-property* $*$ *state*) $\Rightarrow$ *bool*
 where
   is-clique = (λ(*v-set*, p, σ).
     ($\forall$ $v \in$ *v-set*. $v \in$ *observed-non-equivocating-validators* σ
      $\land$ ($\forall$ $v' \in$ *v-set*.
          agreeing (p, (*the-elem* (*L-H-J* σ v)), v')
          $\land$ *later-disagreeing-messages* (p, *the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ

$v))\ v'),\ v',\ \sigma) = \emptyset)))$

lemma (**in** *Protocol*) *non-equivocating-validator-is-non-equivocating-in-past* :
  $\forall\ \sigma\ v\ \sigma'.\ v \in V \wedge \{\sigma, \sigma'\} \subseteq \Sigma \wedge$ *is-future-state* (σ', σ)
  $\longrightarrow v \notin$ *equivocating-validators* σ
  $\longrightarrow v \notin$ *equivocating-validators* σ'
  oops

lemma (**in** *Protocol*) *validator-in-clique-see-L-H-M-of-others-is-singleton* :
  $\forall$ *v-set* $p\ \sigma$. *v-set* $\subseteq V \wedge \sigma \in \Sigma$
  $\longrightarrow$ *is-clique* (*v-set*, p, σ)
  $\longrightarrow (\forall\ v\ v'.\ \{v, v'\} \subseteq$ *v-set* $\longrightarrow$ *is-singleton* (*L-H-M* (*the-elem* (*L-H-J* $\sigma\ v$))
v'))
  sorry

lemma (**in** *Protocol*) *later-from-of-non-sender-not-affected-by-minimal-transitions*
:
  $\forall\ \sigma\ \sigma'\ m\ m'\ v.\ (\sigma, \sigma') \in$ *minimal-transitions* $\wedge\ m \in M$
  $\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$
  $\longrightarrow v \in V - \{$*sender* $m'\}$
  $\longrightarrow$ *later-from* $(m, v, \sigma) =$ *later-from* (m, v, σ')
  apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof−
  fix $\sigma\ \sigma'\ m\ m'\ v$
  assume $(\sigma, \sigma') \in$ *minimal-transitions* $\wedge\ m \in M$
  assume $m' =$ *the-elem* $(\sigma' - \sigma)$
  assume $v \in V - \{$*sender* $m'\}$

  have *later-from* $(m,v,\sigma) = \{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\}$
    by (*simp add*: *later-from-def from-sender-def later-def*)
  also have $\ldots = \{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\} \cup \emptyset$
    by *auto*
  also have $\ldots = \{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\} \cup \{m'' \in \{m'\}.$
sender $m'' = v\}$
  proof−
    have $\{m'' \in \{m'\}.$ *sender* $m'' = v\} = \emptyset$
      using ‹$v \in V - \{$*sender* $m'\}$› **by** *auto*
    thus *?thesis*
      by *blast*
  qed
  also have $\ldots = \{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\} \cup \{m'' \in \{m'\}.$
sender $m'' = v \wedge$ *justified* $m\ m''\}$
  proof−

    have *sender* $m' = v \Longrightarrow$ *justified m* m'
      using ‹$v \in V - \{$*sender* $m'\}$› **by** *auto*
    thus *?thesis*
      by *blast*
  qed
  also have … = $\{m'' \in \sigma \cup \{m'\}$. *sender* $m'' = v \wedge$ *justified m* $m''\}$
    by *auto*
  also have … = $\{m'' \in \sigma'$. *sender* $m'' = v \wedge$ *justified m* $m''\}$
  proof −
    have $\sigma' = \sigma \cup \{m'\}$
      using ‹$(\sigma, \sigma') \in$ *minimal-transitions* $\wedge$ $m \in M$› ‹m' = *the-elem* $(\sigma' - \sigma)$›
minimal-transitions-reconstruction **by** *auto*
    then show *?thesis*
      by *auto*
  qed
  then have … = *later-from* (m,v,σ')
    by (*simp add*: *later-from-def from-sender-def later-def*)
  then show *later-from* (m, v, σ) = *later-from* (m, v, σ')
    using ‹$\{m'' \in \sigma \cup \{m'\}$. *sender* $m'' = v \wedge$ *justified m* $m''\} = \{m'' \in \sigma'$. *sender*
$m'' = v \wedge$ *justified m* $m''\}$› *calculation* **by** *auto*
qed

lemma (**in** *Protocol*) *equivocation-status-of-non-sender-not-affected-by-minimal-transitions*
:
  $\forall$ σ σ' m' v. $(\sigma, \sigma') \in$ *minimal-transitions*
  $\longrightarrow$ m' = *the-elem* $(\sigma' - \sigma)$
  $\longrightarrow$ $v \in V - \{$*sender* $m'\}$
  $\longrightarrow$ $v \in$ *equivocating-validators* σ $\longleftrightarrow$ $v \in$ *equivocating-validators* σ'
  oops

lemma (**in** *Protocol*) *L-M-of-non-sender-not-affected-by-minimal-transitions* :
  $\forall$ σ σ' m' v. $(\sigma, \sigma') \in$ *minimal-transitions*
  $\longrightarrow$ m' = *the-elem* $(\sigma' - \sigma)$
  $\longrightarrow$ $v \in V - \{$*sender* $m'\}$
  $\longrightarrow$ *L-H-M* σ v = *L-H-M* σ' v
  oops

lemma (**in** *Protocol*) *latest-justificationss-of-non-sender-not-affected-by-minimal-transitions*
:
  $\forall$ σ σ' m' v. $(\sigma, \sigma') \in$ *minimal-transitions*
  $\longrightarrow$ m' = *the-elem* $(\sigma' - \sigma)$
  $\longrightarrow$ $v \in V - \{$*sender* $m'\}$
  $\longrightarrow$ *L-H-J* σ v = *L-H-J* σ' v
  oops

lemma (**in** *Protocol*) *later-disagreeing-of-non-sender-not-affected-by-minimal-transitions*
:
  $\forall\ \sigma\ \sigma'\ m\ m'\ v.\ (\sigma, \sigma') \in$ *minimal-transitions* $\wedge\ m \in M$
  $\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$
  $\longrightarrow v \in V - \{$*sender* $m'\}$
  $\longrightarrow$ *later-disagreeing-messages* $(p, m, v, \sigma) =$ *later-disagreeing-messages* $(p, m,$
$v, \sigma')$
  oops

lemma (**in** *Protocol*) *clique-not-affected-by-message-from-non-member* :
  $\forall\ \sigma\ m$ *v-set* $p.\ \sigma \in \Sigma t \wedge m \in M \wedge$ *v-set* $\subseteq V$
  $\longrightarrow$ *immediately-next-message* (σ, m)
  $\longrightarrow$ *sender* $m \notin$ *v-set*
  $\longrightarrow$ *is-clique* (*v-set*, p, σ)
  $\longrightarrow$ *is-clique* (*v-set*, $p, \sigma \cup \{m\}$)
  sorry

lemma (**in** *Protocol*) *free-sub-clique* :
  $\forall\ \sigma\ \sigma'\ m'$ *v-set*. $(\sigma, \sigma') \in$ *minimal-transitions* $\wedge$ *v-set* $\subseteq V$
  $\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$
  $\longrightarrow$ *is-clique* (*v-set*, p, σ) $=$ *is-clique* (*v-set* $- \{$*sender* $m'\}, p, \sigma'$)
  oops

lemma (**in** *Protocol*) *later-messages-from-non-equivocating-validator-include-all-earlier-messages*
:
  $\forall\ v\ \sigma\ \sigma 1\ \sigma 2.\ \sigma \in \Sigma \wedge \sigma 1 \in \Sigma \wedge \sigma 1 \subseteq \sigma \wedge \sigma 2 \subseteq \sigma \wedge \sigma 1 \cap \sigma 2 = \emptyset$
  $\longrightarrow (\forall\ m1 \in \sigma 1.\ sender(m1) = v \longrightarrow (\forall\ m2 \in \sigma 2.\ sender(m2) = v \longrightarrow m1$
$\in justification(m2)))$
  using *strict-subset-of-state-have-immediately-next-messages*
  apply (*simp add*: *immediately-next-message-def*)
  oops

lemma (**in** *Protocol*) *message-between-minimal-transition-is-latest-message* :
  $\forall\ \sigma\ \sigma'\ m'\ v.\ (\sigma, \sigma') \in$ *minimal-transitions*
  $\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$

$\longrightarrow$ *v* $\notin$ *equivocating-validators* σ'
$\longrightarrow$ *m*$'$ = *the-elem* (*L-H-M* σ' *v*)
oops

lemma (**in** *Protocol*) *latest-message-from-non-equivocating-validator-is-previous-latest-or-later*:
$\forall$ σ σ' *m*$'$ *v*. (σ, σ') $\in$ *minimal-transitions*
$\longrightarrow$ *m*$'$ = *the-elem* (σ' − σ)
$\longrightarrow$ *sender m*$'$ $\notin$ *equivocating-validators* σ $\wedge$ *v* $\notin$ *equivocating-validators* σ'
$\longrightarrow$ *the-elem* (*L-H-M* (*justification m*$'$) *v*)
    = *the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ (*sender m*$'$))) *v*)
  $\vee$ *justified* (*the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ (*sender m*$'$))) *v*))
      (*the-elem* (*L-H-M* (*justification m*$'$) *v*))
oops

lemma (**in** *Protocol*) *justified-message-exists-in-later-from*:
$\forall$ σ *m1 m2*. $\sigma \in \Sigma$ $\wedge$ {*m1*, *m2*} $\subseteq$ σ
$\longrightarrow$ *justified m1 m2* $\longrightarrow$ *m2* $\in$ *later-from* (*m1*, *sender m1*, σ)
apply (*simp add*: *later-from-def later-def from-sender-def*)
oops

lemma (**in** *Protocol*) *non-equivocating-message-from-clique-see-clique-agreeing* :
$\forall$ σ σ' *m*$'$ *v-set*. (σ, σ') $\in$ *minimal-transitions* $\wedge$ *v-set* $\subseteq$ *V*
$\longrightarrow$ *m*$'$ = *the-elem* (σ' − σ)
$\longrightarrow$ *is-clique* (*v-set*, *p*, σ) $\wedge$ *sender m*$'$ $\in$ *v-set* $\wedge$ *sender m*$'$ $\notin$ *equivocating-validators*
σ'
$\longrightarrow$ *v-set* $\subseteq$ *agreeing-validators* (*p*, *justification m*$'$)
oops

lemma (**in** *Protocol*) *new-message-from-majority-clique-see-members-agreeing* :
$\forall$ σ σ' *m*$'$ *v-set*. (σ, σ') $\in$ *minimal-transitions* $\wedge$ *v-set* $\subseteq$ *V*
$\longrightarrow$ *m*$'$ = *the-elem* (σ' − σ)
$\longrightarrow$ *is-clique* (*v-set*, *p*, σ) $\wedge$ *sender m*$'$ $\in$ *v-set* $\wedge$ *sender m*$'$ $\notin$ *equivocating-validators*
σ'
  $\wedge$ ($\forall$ *v* $\in$ *v-set*. *majority* (*v-set*, *the-elem* (*L-H-J* σ *v*)))
$\longrightarrow$ *sender m*$'$ $\in$ *agreeing-validators* (*p*, *justification m*$'$)
oops

lemma (**in** *Protocol*) *latest-message-in-justification-of-new-message-is-latest-message* :
$\forall$ σ σ' m' *v-set*. $(\sigma, \sigma') \in$ *minimal-transitions* $\wedge$ *v-set* $\subseteq V$
$\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$
$\longrightarrow$ *sender* $m' \notin$ *equivocating-validators* σ'
$\longrightarrow$ *the-elem* (*L-H-M* (*justification* m') (*sender* m')) = *the-elem* (*L-H-M* σ (*sender* m'))
oops

lemma (**in** *Protocol*) *latest-message-justified-by-new-message* :
$\forall$ σ σ' m' *v-set*. $(\sigma, \sigma') \in$ *minimal-transitions* $\wedge$ *v-set* $\subseteq V$
$\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$
$\longrightarrow$ *sender* $m' \notin$ *equivocating-validators* σ'
$\longrightarrow$ *justified* (*the-elem* (*L-H-M* σ (*sender* m'))) m'
oops

lemma (**in** *Protocol*) *nothing-later-than-latest-honest-message* :
$\forall$ v σ m. $v \in V \wedge \sigma \in \Sigma \wedge m \in M$
$\longrightarrow v \notin$ *equivocating-validators* σ'
$\longrightarrow$ *later-from* (*the-elem* (*L-H-M* σ v), v, σ) = $\emptyset$
oops

lemma (**in** *Protocol*) *later-messages-for-sender-is-new-message* :
$\forall$ σ σ' m' *v-set*. $(\sigma, \sigma') \in$ *minimal-transitions* $\wedge$ *v-set* $\subseteq V$
$\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$
$\longrightarrow$ *sender* $m' \notin$ *equivocating-validators* σ'
$\longrightarrow$ *later-from* (*the-elem* (*L-H-M* σ (*sender* m')), *sender* m', σ') = $\{m'\}$
oops

lemma (**in** *Protocol*) *later-disagreeing-is-monotonic*:
$\forall$ v σ *m1* *m2*. $v \in V \wedge \sigma \in \Sigma \wedge \{m1, m2\} \subseteq M$
$\longrightarrow$ *justified m1 m2*
$\longrightarrow$ *later-disagreeing-messages* (p, *m2*, v, σ) $\subseteq$ *later-disagreeing-messages* (p, *m1*, v, σ)
oops

lemma (**in** *Protocol*) *empty-later-disagreeing-messages-in-new-message* :
$\forall$ σ σ' m' *v-set* v p. $(\sigma, \sigma') \in$ *minimal-transitions* $\wedge$ *v-set* $\subseteq V \wedge v \in V$
$\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$
$\longrightarrow$ *sender* $m' \notin$ *equivocating-validators* σ'
$\longrightarrow v \notin$ *equivocating-validators* σ
$\longrightarrow$ *later-disagreeing-messages* (p, (*the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ (*sender*

m′))) *v*)), *v*, σ) = ∅
  ⟶ *later-disagreeing-messages* (*p*, (*the-elem* (*L-H-M* (*justification m′*) *v*)), *v*, σ)
= ∅
  oops

lemma (**in** *Protocol*) *clique-not-affected-by-honest-message-from-member* :
  ∀ σ *m v-set p*. σ ∈ Σ*t* ∧ *m* ∈ *M* ∧ *v-set* ⊆ *V*
  ⟶ *majority-driven p*
  ⟶ *immediately-next-message* (σ, *m*)
  ⟶ *sender m* ∈ *v-set*
  ⟶ ¬ *is-equivocating* (σ ∪ {*m*}) (*sender m*)
  ⟶ *is-clique* (*v-set*, *p*, σ)
  ⟶ *is-clique* (*v-set*, *p*, σ ∪ {*m*})
  sorry

definition (**in** *Params*) *gt-threshold* :: (*validator set* ∗ *state*) ⇒ *bool*
  where
    gt-threshold
        = (λ(*v-set*, σ).(*weight-measure v-set* > (*weight-measure V*) *div 2* + *t* − *weight-measure* (*equivocating-validators* σ)))

lemma (**in** *Protocol*) *gt-threshold-imps-majority-for-any-validator* :
  ∀ σ *v-set p*. σ ∈ Σ ∧ *v-set* ⊆ *V*
  ⟶ *gt-threshold* (*v-set*, σ)
  ⟶ (∀ *v* ∈ *v-set*. *majority* (*v-set*, *the-elem* (*L-H-J* σ *v*)))
  oops

definition (**in** *Params*) *is-clique-oracle* :: (*validator set* ∗ *state* ∗ *consensus-value-property*) ⇒ *bool*
  where
    is-clique-oracle
       = (λ(*v-set*, σ, *p*). (*is-clique* (*v-set*, *p*, σ) ∧ *gt-threshold* (*v-set*, σ)))

lemma (**in** *Protocol*) *clique-oracles-preserved-over-message-from-non-member* :
  ∀ σ *m v-set p*. σ ∈ Σ*t* ∧ *m* ∈ *M* ∧ *v-set* ⊆ *V*
  ⟶ *majority-driven p*
  ⟶ *immediately-next-message* (σ, *m*)
  ⟶ *sender m* ∉ *v-set*

```
  ⟶ is-clique-oracle (v-set, σ, p)
  ⟶ is-clique-oracle (v-set, σ ∪ {m}, p)
  using clique-not-affected-by-message-from-non-member
  unfolding is-clique-oracle-def gt-threshold-def
  using equivocation-fault-weight-is-monotonic
  apply auto
 by (smt Un-insert-right Σt-is-subset-of-Σ equivocation-fault-weight-def state-transition-by-immediately-next-m
subsetCE subset-insertI sup-bot.right-neutral)


lemma (in Protocol) clique-oracles-preserved-over-message-from-non-equivocating-member
:
  ∀ σ m v-set p. σ ∈ Σt ∧ m ∈ M ∧ v-set ⊆ V
  ⟶ majority-driven p
  ⟶ immediately-next-message (σ, m)
  ⟶ sender m ∈ v-set
  ⟶ ¬ is-equivocating (σ ∪ {m}) (sender m)
  ⟶ is-clique-oracle (v-set, σ, p)
  ⟶ is-clique-oracle (v-set, σ ∪ {m}, p)
  using clique-not-affected-by-honest-message-from-member
  unfolding is-clique-oracle-def gt-threshold-def
  using equivocating-validators-preserved-over-honest-message
  using Σt-is-subset-of-Σ
  sorry


lemma (in Protocol) clique-oracles-preserved-over-message-from-equivocating-member
:
  ∀ σ m v-set p. σ ∈ Σt ∧ m ∈ M ∧ v-set ⊆ V
  ⟶ majority-driven p
  ⟶ immediately-next-message (σ, m)
  ⟶ sender m ∈ v-set
  ⟶ is-equivocating (σ ∪ {m}) (sender m)
  ⟶ σ ∪ {m} ∈ Σt
  ⟶ is-clique-oracle (v-set, σ, p)
  ⟶ is-clique-oracle (v-set, σ ∪ {m}, p)

  sorry



lemma (in Protocol) clique-oracles-preserved-over-immediately-next-message :
  ∀ σ m v-set p. σ ∈ Σt ∧ v-set ⊆ V
  ⟶ majority-driven p
  ⟶ immediately-next-message (σ, m)
  ⟶ σ ∪ {m} ∈ Σt
  ⟶ is-clique-oracle (v-set, σ, p)
  ⟶ is-clique-oracle (v-set, σ ∪ {m}, p)
  using clique-oracles-preserved-over-message-from-non-member
```

```
        clique-oracles-preserved-over-message-from-non-equivocating-member
        clique-oracles-preserved-over-message-from-equivocating-member
  by (metis (no-types, lifting) Un-insert-right Σt-def insert-subset mem-Collect-eq
state-is-subset-of-M)



lemma (in Protocol) clique-imps-everyone-agreeing :
  ∀ σ v-set p. σ ∈ Σ ∧ v-set ⊆ V
  ⟶ is-clique (v-set, p, σ)
  ⟶ v-set ⊆ agreeing-validators (p, σ)
  apply (rule, rule, rule, rule, rule)
proof−
  fix σ v-set p assume σ ∈ Σ ∧ v-set ⊆ V  and is-clique (v-set, p, σ)
  then have clique: ∀ v ∈ v-set. v ∈ observed-non-equivocating-validators σ
          ∧ later-disagreeing-messages (p,
                                       the-elem (L-H-M
                                          (the-elem (L-H-J σ v)) v)
                                       , v, σ) = ∅
    by (simp add: is-clique-def)
  then have p-on-est : ∀ v ∈ v-set. (∀ m ∈ {m' ∈ σ. sender m' = v
                                      ∧ justified (the-elem (L-H-M
                                                   (the-elem (L-H-J σ v)) v))
                                                 m'}.
                                       p(est m))
   by (simp add: later-disagreeing-messages-def later-from-def later-def from-sender-def)
  have ∀ v ∈ v-set. v ∈ observed-non-equivocating-validators σ
    using clique by simp
  then have ∀ v ∈ v-set. the-elem (L-H-J σ v)
                   = justification (the-elem (L-H-M σ v))
    apply (simp add: L-H-J-def)
   by (metis ‹σ ∈ Σ ∧ v-set ⊆ V› empty-iff is-singleton-the-elem L-H-M-of-observed-non-equivocating-validator-
singletonD singletonI the-elem-image-unique)
  then have justified-ok: ∀ v ∈ v-set. justified (the-elem (L-H-M
                                                   (the-elem (L-H-J σ v)) v))
                                     (the-elem (L-H-M σ v))
    using validator-in-clique-see-L-H-M-of-others-is-singleton
    by (smt Diff-iff L-H-M-def L-H-M-is-in-the-state L-M-from-non-observed-validator-is-empty
M-type ‹∀ v∈v-set. v ∈ observed-non-equivocating-validators σ› ‹σ ∈ Σ ∧ v-set ⊆ V›
‹is-clique (v-set, p, σ)› empty-subsetI insert-subset is-singleton-the-elem justified-def
observed-non-equivocating-validators-def state-is-subset-of-M subsetCE)
  have sender-ok: ∀ v ∈ v-set. sender (the-elem (L-H-M σ v)) = v
   using ‹∀ v ∈ v-set. v ∈ observed-non-equivocating-validators σ› sender-of-L-H-M
    using ‹σ ∈ Σ ∧ v-set ⊆ V› by blast
  have ∀ v ∈ v-set.  the-elem (L-H-M σ v) ∈ σ
   using ‹∀ v ∈ v-set. v ∈ observed-non-equivocating-validators σ› L-H-M-is-in-the-state
    using ‹σ ∈ Σ ∧ v-set ⊆ V› by blast
  then have ∀ v ∈ v-set. p (est (the-elem (L-H-M σ v)))
    using p-on-est sender-ok justified-ok
    by blast
  then have ∀ v ∈ v-set. p (the-elem (L-H-E σ v))
    apply (simp add: L-H-E-def)
   by (metis (no-types, lifting) ‹∀ v∈v-set. v ∈ observed-non-equivocating-validators
σ› ‹σ ∈ Σ ∧ v-set ⊆ V› empty-iff is-singleton-the-elem L-H-M-of-observed-non-equivocating-validator-is-singleto
singletonD singletonI the-elem-image-unique)
  then show v-set ⊆ agreeing-validators (p, σ)
    unfolding agreeing-validators-def agreeing-def
   by (smt ‹∀ v∈v-set. v ∈ observed-non-equivocating-validators σ› ‹σ ∈ Σ ∧ v-set ⊆
V› is-singleton-the-elem mem-Collect-eq L-H-E-of-observed-non-equivocating-validator-is-singleton
old.prod.case singletonD subsetI)
qed


lemma (in Protocol) threshold-sized-clique-imps-estimator-agreeing :
  ∀ σ v-set p. σ ∈ Σt ∧ v-set ⊆ V
  ⟶ finite v-set
  ⟶ majority-driven p
  ⟶ is-clique (v-set − equivocating-validators σ, p, σ) ∧ gt-threshold (v-set −
equivocating-validators σ, σ)
  ⟶ (∀ c ∈ ε σ. p c)
  apply (rule, rule, rule, rule, rule, rule, rule, rule)
proof −
  fix σ v-set p c
  assume σ ∈ Σt ∧ v-set ⊆ V
  and finite v-set
  and majority-driven p
  and is-clique (v-set − equivocating-validators σ, p, σ) ∧ gt-threshold (v-set −
equivocating-validators σ, σ)
  and c ∈ ε σ
  then have v-set − equivocating-validators σ ⊆ agreeing-validators (p, σ)
    using clique-imps-everyone-agreeing
    by (meson Diff-subset Σt-is-subset-of-Σ subsetCE subset-trans)
  then have weight-measure (v-set − equivocating-validators σ) ≤ weight-measure
(agreeing-validators (p, σ))
   using agreeing-validators-finite equivocating-validators-def weight-measure-subset-gte
        Σt-is-subset-of-Σ ‹σ ∈ Σt ∧ v-set ⊆ V› ‹finite v-set›
    by (simp add: Σt-def agreeing-validators-type)
  have weight-measure (v-set − equivocating-validators σ) > (weight-measure V)
div 2 + t − weight-measure (equivocating-validators σ)
    using ‹is-clique (v-set − equivocating-validators σ, p, σ) ∧ gt-threshold (v-set
− equivocating-validators σ, σ)›
    unfolding gt-threshold-def by simp
  then have weight-measure (v-set − equivocating-validators σ) > (weight-measure
V) div 2
   using Σt-def ‹σ ∈ Σt ∧ v-set ⊆ V› equivocation-fault-weight-def is-faults-lt-threshold-def

    by auto
  then have weight-measure (v-set − equivocating-validators σ) > (weight-measure
```

(*V* − *equivocating-validators* σ)) *div 2*
  proof −
    have *finite* (*V* − *equivocating-validators* σ)
      using *V-type equivocating-validators-is-finite*
      by *simp*
    moreover have *V* − *equivocating-validators* σ ⊆ *V*
      by (*simp add*: *Diff-subset*)
   ultimately have (*weight-measure V*) *div 2* ≥ (*weight-measure* (*V* − *equivocating-validators* σ)) *div 2*
      using *weight-measure-subset-gte*
      by (*simp add*: *V-type*)
    then show *?thesis*
     using ‹*weight-measure V* / *2* < *weight-measure* (*v-set* − *equivocating-validators* σ)› **by** *linarith*
  qed
  then have *weight-measure* (*agreeing-validators* (*p*, σ)) > *weight-measure* (*V* − *equivocating-validators* σ) *div 2*
    using ‹*weight-measure* (*v-set* − *equivocating-validators* σ) ≤ *weight-measure* (*agreeing-validators* (*p*, σ))›
    by *linarith*
  then show *p c*
   using ‹*majority-driven p*› **unfolding** *majority-driven-def majority-def gt-threshold-def*
    using ‹*c* ∈ ε σ›
   using *Mi.simps* Σ*t-is-subset-of-*Σ ‹σ ∈ Σ*t* ∧ *v-set* ⊆ *V*› *non-justifying-message-exists-in-M-0* **by** *blast*
qed

lemma (**in** *Protocol*) *clique-oracle-for-all-futures* :
  ∀ σ *v-set p*. σ ∈ Σ*t* ∧ *v-set* ⊆ *V*
  ⟶ *majority-driven p*
  ⟶ *is-clique-oracle* (*v-set*, σ, *p*)
  ⟶ (∀ σ′ ∈ *futures* σ. *is-clique-oracle* (*v-set*, σ′, *p*))
  apply (*rule*+)
proof −
  fix σ *v-set p* σ′
  assume σ ∈ Σ*t* ∧ *v-set* ⊆ *V* **and** *majority-driven p* **and** *is-clique-oracle* (*v-set*, σ, *p*) **and** σ′ ∈ *futures* σ
  show *is-clique-oracle* (*v-set*, σ′, *p*)
    using *clique-oracles-preserved-over-immediately-next-message*

    sorry
qed

lemma (**in** *Protocol*) *clique-oracle-is-safety-oracle* :
  ∀ σ *v-set p*. σ ∈ Σ*t* ∧ *v-set* ⊆ *V*
  ⟶ *finite v-set*
  ⟶ *majority-driven p*

$\longrightarrow$ *is-clique-oracle* (*v-set*, σ, *p*)
$\longrightarrow$ ($\forall$ σ' $\in$ *futures* σ. *naturally-corresponding-state-property p* σ')
apply *rule+*
proof −
fix σ *v-set p* σ'
assume $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$ **and** *finite v-set* **and** *majority-driven p* **and** *is-clique-oracle* (*v-set*, σ, *p*) **and** $\sigma' \in$ *futures* σ
then have $\forall$ σ' $\in$ *futures* σ. *is-clique-oracle* (*v-set*, σ', *p*)
using *clique-oracle-for-all-futures*
by *blast*
then have $\forall$ σ' $\in$ *futures* σ. $\forall$ $c \in \varepsilon$ σ'. *p c*
using ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› ‹*finite v-set*› ‹*majority-driven p*› ‹$\sigma' \in$ *futures* σ›
using *threshold-sized-clique-imps-estimator-agreeing*
apply (*simp add*: *futures-def is-clique-oracle-def*)
sorry
then show *naturally-corresponding-state-property p* σ'
apply (*simp add*: *naturally-corresponding-state-property-def*)
using ‹$\sigma' \in$ *futures* σ› **by** *blast*
qed

end
theory *Inspector*

imports *Main CBCCasper LatestMessage StateTransition ConsensusSafety*

begin

definition *agreeing* :: (*consensus-value-property* $*$ *state* $*$ *validator*) $\Rightarrow$ *bool*
where
agreeing = (λ(*p*, σ, *v*). $\forall$ $c \in$ *L-H-E* σ *v*. *p c*)

definition *agreeing-validators* :: (*consensus-value-property* ∗ *state*) ⇒ *validator set*
where
agreeing-validators = (λ(*p*, σ).{*v* ∈ *observed-non-equivocating-validators* σ. *agreeing* (*p*, σ, *v*)})

lemma (**in** *Protocol*) *agreeing-validators-type* :
∀ σ ∈ Σ. *agreeing-validators* (*p*, σ) ⊆ *V*
apply (*simp add*: *observed-non-equivocating-validators-def agreeing-validators-def*)
using *observed-type-for-state* **by** *auto*

lemma (**in** *Protocol*) *agreeing-validators-finite* :
∀ σ ∈ Σ. *finite* (*agreeing-validators* (*p*, σ))
by (*meson V-type agreeing-validators-type rev-finite-subset*)

lemma (**in** *Protocol*) *agreeing-validators-are-observed-non-equivocating-validators* :
∀ σ ∈ Σ. *agreeing-validators* (*p*, σ) ⊆ *observed-non-equivocating-validators* σ
by (*simp add*: *agreeing-validators-def*)

lemma (**in** *Protocol*) *agreeing-validators-are-not-equivocating* :
∀ σ ∈ Σ. *agreeing-validators* (*p*, σ) ∩ *equivocating-validators* σ = ∅
using *agreeing-validators-are-observed-non-equivocating-validators*
observed-non-equivocating-validators-are-not-equivocating
by *blast*

definition (**in** *Params*) *disagreeing-validators* :: (*consensus-value-property* ∗ *state*) ⇒ *validator set*
where
disagreeing-validators = (λ(*p*, σ). *V* − *agreeing-validators* (*p*, σ) − *equivocating-validators* σ)

lemma (**in** *Protocol*) *disagreeing-validators-type* :
∀ σ ∈ Σ. *disagreeing-validators* (*p*, σ) ⊆ *V*
apply (*simp add*: *disagreeing-validators-def*)
by *auto*

lemma (**in** *Protocol*) *disagreeing-validators-are-non-observed-or-not-agreeing* :
∀ σ ∈ Σ. *disagreeing-validators* (*p*, σ) = {*v* ∈ *V* − *equivocating-validators* σ. *v* ∉ *observed* σ ∨ (∃ *c* ∈ *L-H-E* σ *v*. ¬ *p c*)}
apply (*simp add*: *disagreeing-validators-def agreeing-validators-def observed-non-equivocating-validators-def agreeing-def*)
by *blast*

lemma (**in** *Protocol*) *disagreeing-validators-include-not-agreeing-validators* :
∀ σ ∈ Σ. {*v* ∈ *V* − *equivocating-validators* σ. ∃ *c* ∈ *L-H-E* σ *v*. ¬ *p c*} ⊆ *disagreeing-validators* (*p*, σ)
using *disagreeing-validators-are-non-observed-or-not-agreeing* **by** *blast*

lemma (**in** *Protocol*) *weight-measure-agreeing-plus-equivocating* :
$\forall\ \sigma \in \Sigma.$ *weight-measure* (*agreeing-validators* (p, σ) $\cup$ *equivocating-validators* σ)
= *weight-measure* (*agreeing-validators* (p, σ)) + *equivocation-fault-weight* σ
unfolding *equivocation-fault-weight-def*
using *agreeing-validators-are-not-equivocating weight-measure-disjoint-plus agreeing-validators-finite equivocating-validators-is-finite*
by *simp*

lemma (**in** *Protocol*) *disagreeing-validators-weight-combined* :
$\forall\ \sigma \in \Sigma.$ *weight-measure* (*disagreeing-validators* (p, σ)) = *weight-measure* V −
weight-measure (*agreeing-validators* (p, σ)) − *equivocation-fault-weight* σ
unfolding *disagreeing-validators-def*
using *weight-measure-agreeing-plus-equivocating*
unfolding *equivocation-fault-weight-def*
using *agreeing-validators-are-not-equivocating weight-measure-subset-minus agreeing-validators-finite equivocating-validators-is-finite*
by (*smt Diff-empty Diff-iff Int-iff V-type agreeing-validators-type equivocating-validators-type finite-Diff old.prod.case subset-iff*)

lemma (**in** *Protocol*) *agreeing-validators-weight-combined* :
$\forall\ \sigma \in \Sigma.$ *weight-measure* (*agreeing-validators* (p, σ)) = *weight-measure* V −
weight-measure (*disagreeing-validators* (p, σ)) − *equivocation-fault-weight* σ
using *disagreeing-validators-weight-combined*
by *simp*

definition (**in** *Params*) *majority* :: (*validator set* ∗ *state*) ⇒ *bool*
where
majority = (λ(*v-set*, σ). (*weight-measure v-set* > (*weight-measure* (V − *equivocating-validators* σ)) *div 2*))

definition (**in** *Protocol*) *majority-driven* :: *consensus-value-property* ⇒ *bool*
where
majority-driven p = ($\forall\ \sigma \in \Sigma.$ *majority* (*agreeing-validators* (p, σ), σ) $\longrightarrow$ ($\forall$
$c \in \varepsilon\ \sigma.\ p\ c$))

definition (**in** *Protocol*) *max-driven* :: *consensus-value-property* ⇒ *bool*
where
max-driven p =
($\forall\ \sigma \in \Sigma.$ *weight-measure* (*agreeing-validators* (p, σ)) > *weight-measure*
(*disagreeing-validators* (p, σ)) $\longrightarrow$ ($\forall\ c \in \varepsilon\ \sigma.\ p\ c$))

definition (**in** *Protocol*) *max-driven-for-future* :: *consensus-value-property* ⇒ *state*
⇒ *bool*
where
max-driven-for-future p σ =

$(\forall\ \sigma' \in \Sigma.\ \textit{is-future-state}\ (\sigma,\ \sigma')$
$\longrightarrow \textit{weight-measure}\ (\textit{agreeing-validators}\ (p, \sigma')) > \textit{weight-measure}\ (\textit{disagreeing-validators}\ (p,\ \sigma')) \longrightarrow (\forall\ \ c \in \varepsilon\ \sigma'.\ p\ c))$

definition *later-disagreeing-messages* :: (*consensus-value-property* ∗ *message* ∗ *validator* ∗ *state*) ⇒ *message set*
where
$\textit{later-disagreeing-messages} = (\lambda(p,\ m,\ v,\ \sigma).\{m' \in \textit{later-from}\ (m,\ v,\ \sigma).\ \neg\ p\ (\textit{est}\ m')\})$

lemma (**in** *Protocol*) *later-disagreeing-messages-type* :
$\forall\ \ p\ \sigma\ v\ m.\ \sigma \in \Sigma \wedge v \in V \wedge m \in M \longrightarrow \textit{later-disagreeing-messages}\ (p,\ m,\ v,\ \sigma) \subseteq M$
unfolding *later-disagreeing-messages-def*
using *later-from-type-for-state* **by** *auto*

lemma (**in** *Protocol*) *non-equivocating-validator-is-non-equivocating-in-past* :
$\forall\ \ \sigma\ v\ \sigma'.\ v \in V \wedge \{\sigma,\ \sigma'\} \subseteq \Sigma \wedge \textit{is-future-state}\ (\sigma',\ \sigma)$
$\longrightarrow v \notin \textit{equivocating-validators}\ \sigma$
$\longrightarrow v \notin \textit{equivocating-validators}\ \sigma'$

oops

definition (**in** *Params*) *gt-threshold* :: (*validator set* ∗ *state*) ⇒ *bool*
where
gt-threshold
$= (\lambda(\textit{v-set},\ \sigma).(\textit{weight-measure v-set} > (\textit{weight-measure}\ V)\ \textit{div 2} + t\ \textit{div 2} - \textit{weight-measure}\ (\textit{equivocating-validators}\ \sigma)))$

lemma (**in** *Protocol*) *gt-threshold-imps-majority-for-any-validator* :
$\forall\ \ \sigma\ \textit{v-set}\ p.\ \sigma \in \Sigma \wedge \textit{v-set} \subseteq V$
$\longrightarrow \textit{gt-threshold}\ (\textit{v-set},\ \sigma)$
$\longrightarrow (\forall\ \ v \in \textit{v-set}.\ \textit{majority}\ (\textit{v-set},\ \textit{the-elem}\ (\textit{L-H-J}\ \sigma\ v)))$
oops

definition (**in** *Params*) *inspector* :: (*validator set* ∗ *state* ∗ *consensus-value-property*) ⇒ *bool*
where
inspector
$= (\lambda(\textit{v-set},\ \sigma,\ p).\ \textit{v-set} \neq \emptyset\ \wedge$
$(\forall\ \ v \in \textit{v-set}.\ v \in \textit{agreeing-validators}\ (p,\ \sigma)$
$\wedge\ (\exists\ \ \textit{v-set}'.\ \textit{v-set}' \subseteq \textit{v-set} \wedge \textit{gt-threshold}(\textit{v-set}',\ \textit{the-elem}\ (\textit{L-H-J}\ \sigma\ v))$

$\wedge$ ($\forall$ v' $\in$ *v-set'*.
v' $\in$ *agreeing-validators* (*p*, (*the-elem* (*L-H-J* σ *v*)))
$\wedge$ *later-disagreeing-messages* (*p*, *the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ *v*)) v'), v', σ) = $\emptyset$))))

lemma (**in** *Protocol*) *validator-in-inspector-see-L-H-M-of-others-is-singleton* :
$\forall$ *v-set p* σ. *v-set* $\subseteq$ *V* $\wedge$ $\sigma \in \Sigma$
$\longrightarrow$ *inspector* (*v-set*, σ, *p*)
$\longrightarrow$ ($\forall$ *v* v'. {*v*, v'} $\subseteq$ *v-set* $\longrightarrow$ *is-singleton* (*L-H-M* (*the-elem* (*L-H-J* σ *v*)) v'))
oops

lemma (**in** *Protocol*) *inspector-imps-everyone-observed-non-equivocating* :
$\forall$ σ *v-set p*. $\sigma \in \Sigma$ $\wedge$ *v-set* $\subseteq$ *V*
$\longrightarrow$ *inspector* (*v-set*, σ, *p*)
$\longrightarrow$ *v-set* $\subseteq$ *observed-non-equivocating-validators* (σ)
apply (*simp add*: *inspector-def agreeing-validators-def*)
by *blast*

lemma (**in** *Protocol*) *inspector-imps-everyone-agreeing* :
$\forall$ σ *v-set p*. $\sigma \in \Sigma$ $\wedge$ *v-set* $\subseteq$ *V*
$\longrightarrow$ *inspector* (*v-set*, σ, *p*)
$\longrightarrow$ *v-set* $\subseteq$ *agreeing-validators* (*p*, σ)
apply (*simp add*: *inspector-def*)
by *blast*

lemma (**in** *Protocol*) *inspector-imps-gt-threshold* :
$\forall$ σ *v-set p*. $\sigma \in \Sigma$ $\wedge$ *v-set* $\subseteq$ *V*
$\longrightarrow$ *inspector* (*v-set*, σ, *p*)
$\longrightarrow$ *gt-threshold*(*v-set*, σ)
apply (*rule*+)
proof –
fix σ *v-set p*
assume $\sigma \in \Sigma$ $\wedge$ *v-set* $\subseteq$ *V*
assume *inspector* (*v-set*, σ, *p*)
hence $\exists$ *v* $\in$ *v-set*. $\exists$ *v-set'*. *v-set'* $\subseteq$ *v-set* $\wedge$ *gt-threshold*(*v-set'*, *the-elem* (*L-H-J* σ *v*))
apply (*simp add*: *inspector-def*)
by *blast*
hence $\exists$ *v* $\in$ *v-set*. *gt-threshold*(*v-set*, *the-elem* (*L-H-J* σ *v*))
apply (*simp add*: *gt-threshold-def*)
using *weight-measure-subset-gte*
by (*smt* ‹$\sigma \in \Sigma$ $\wedge$ *v-set* $\subseteq$ *V*›)
obtain *v* **where** *v* $\in$ *v-set* $\wedge$ *gt-threshold*(*v-set*, *the-elem* (*L-H-J* σ *v*))
using ‹$\exists$ *v*$\in$*v-set*. *gt-threshold* (*v-set*, *the-elem* (*L-H-J* σ *v*))› **by** *blast*
hence $\forall$ σ' $\in$ *L-H-J* σ *v*. $\sigma' \subseteq \sigma$
using *L-H-J-is-subset-of-the-state* ‹$\sigma \in \Sigma$ $\wedge$ *v-set* $\subseteq$ *V*›
by *blast*

```
  hence is-singleton (L-H-J σ v) ∧ (∀ σ' ∈ L-H-J σ v. σ' ⊆ σ)
   using L-H-J-is-subset-of-the-state ‹σ ∈ Σ ∧ v-set ⊆ V› L-H-J-of-observed-non-equivocating-validator-is-singl
        ‹inspector (v-set, σ, p)›
    apply (simp add: inspector-def agreeing-validators-def)
    using ‹v ∈ v-set ∧ gt-threshold (v-set, the-elem (L-H-J σ v))› by auto
  hence the-elem (L-H-J σ v) ⊆ σ
    by (metis insert-iff is-singleton-the-elem)
  then show gt-threshold (v-set, σ)
    using ‹v ∈ v-set ∧ gt-threshold(v-set, the-elem (L-H-J σ v))›
    apply (simp add: gt-threshold-def)
    using equivocation-fault-weight-is-monotonic
    apply (simp add: equivocation-fault-weight-def)
   by (smt L-H-J-type ‹σ ∈ Σ ∧ v-set ⊆ V› ‹is-singleton (L-H-J σ v) ∧ (∀σ'∈L-H-J
σ v. σ' ⊆ σ)› is-singleton-the-elem singletonI subsetCE)
qed


lemma (in Protocol) gt-threshold-imps-estimator-agreeing :
  ∀ σ v-set p. σ ∈ Σt ∧ v-set ⊆ V
  ⟶ finite v-set
  ⟶ majority-driven p
  ⟶ v-set ⊆ agreeing-validators (p, σ)
  ⟶ gt-threshold (v-set, σ)
  ⟶ (∀ c ∈ ε σ. p c)
  apply (rule, rule, rule, rule, rule, rule, rule, rule, rule)
proof -
  fix σ v-set p c
 assume σ ∈ Σt ∧ v-set ⊆ V finite v-set majority-driven p v-set ⊆ agreeing-validators
(p, σ) gt-threshold (v-set, σ) c ∈ ε σ
  then have weight-measure v-set ≤ weight-measure (agreeing-validators (p, σ))
    using inspector-imps-everyone-agreeing
        weight-measure-subset-gte
        Σt-is-subset-of-Σ agreeing-validators-type by auto
   then have weight-measure v-set > (weight-measure V) div 2 + t div 2 −
weight-measure (equivocating-validators σ)
    using ‹σ ∈ Σt ∧ v-set ⊆ V› ‹gt-threshold (v-set, σ)›
        gt-threshold-def
        Σt-is-subset-of-Σ by auto
  then have weight-measure v-set > (weight-measure V) div 2 − weight-measure
(equivocating-validators σ) div 2
   using Σt-def ‹σ ∈ Σt ∧ v-set ⊆ V› equivocation-fault-weight-def is-faults-lt-threshold-def
    by auto
 then have weight-measure v-set > (weight-measure (V − equivocating-validators
σ)) div 2
     by (metis Protocol.V-type Protocol-axioms Σt-is-subset-of-Σ ‹σ ∈ Σt ∧ v-set
⊆ V› diff-divide-distrib equivocating-validators-is-finite equivocating-validators-type
subsetCE weight-measure-subset-minus)
  then have weight-measure (agreeing-validators (p, σ)) > weight-measure (V −
equivocating-validators σ) div 2
```

using ‹*weight-measure v-set* $\leq$ *weight-measure* (*agreeing-validators* (*p*, σ))›
by *auto*
then show *p c*
using ‹*majority-driven p*› **unfolding** *majority-driven-def majority-def gt-threshold-def*
using ‹*c* $\in \varepsilon\ \sigma$› *Mi.simps* Σ*t-is-subset-of-*Σ ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› *non-justifying-message-exists-in-M-0*
by *blast*
qed

lemma (**in** *Protocol*) *inspector-imps-estimator-agreeing* :
$\forall$ σ *v-set p*. $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *finite v-set*
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *inspector* (*v-set*, σ, *p*)
$\longrightarrow$ ($\forall$ *c* $\in \varepsilon\ \sigma$. *p c*)
by (*simp add*: *gt-threshold-imps-estimator-agreeing inspector-imps-gt-threshold* Σ*t-def inspector-imps-everyone-agreeing*)

lemma (**in** *Protocol*) *later-from-of-non-sender-not-affected-by-minimal-transitions*
:
$\forall$ σ *m m' v*. $\sigma \in \Sigma \wedge m \in M \wedge m' \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, *m'*)
$\longrightarrow$ $v \in V -$ {*sender m'*}
$\longrightarrow$ *later-from* (*m*, *v*, σ) = *later-from* (*m*, *v*, $\sigma \cup$ {*m'*})
apply (*simp add*: *later-from-def*)
by *auto*

lemma (**in** *Protocol*) *from-sender-of-non-sender-not-affected-by-minimal-transitions*
:
$\forall$ σ *m m' v*. $\sigma \in \Sigma \wedge m \in M \wedge m' \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, *m'*)
$\longrightarrow$ $v \in V -$ {*sender m'*}
$\longrightarrow$ *from-sender* (*v*, σ) = *from-sender* (*v*, $\sigma \cup$ {*m'*})
apply (*simp add*: *from-sender-def*)
by *auto*

lemma (**in** *Protocol*) *equivocation-status-of-non-sender-not-affected-by-minimal-transitions*
:
$\forall$ σ *m v*. $\sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, *m*)
$\longrightarrow$ $v \in V -$ {*sender m*}
$\longrightarrow$ $v \in$ *equivocating-validators* σ $\longleftrightarrow$ $v \in$ *equivocating-validators* ($\sigma \cup$ {*m*})

apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix σ *m v*
assume $\sigma \in \Sigma \wedge m \in M \wedge v \in V$
and *immediately-next-message* (σ, *m*)
and $v \in V - \{$*sender m*$\}$
then have *g1*: *observed* $\sigma \subseteq$ *observed* ($\sigma \cup \{m\}$)
apply (*simp add*: *observed-def*)
by *auto*
have *g2*: *is-equivocating* σ *v* = *is-equivocating* ($\sigma \cup \{m\}$) *v*
using ‹$v \in V - \{$*sender m*$\}$›
apply (*simp add*: *is-equivocating-def equivocation-def*)
by *blast*
show ($v \in$ *equivocating-validators* σ) = ($v \in$ *equivocating-validators* ($\sigma \cup \{m\}$))
apply (*simp add*: *equivocating-validators-def*)
using *g1 g2*
by (*metis* (*mono-tags*, *lifting*) *Un-insert-right is-equivocating-def mem-Collect-eq observed-def sup-bot.right-neutral*)
qed

lemma (**in** *Protocol*) *L-H-M-of-non-sender-not-affected-by-minimal-transitions* :
$\forall$ σ *m v*. $\sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, *m*)
$\longrightarrow$ $v \in V - \{$*sender m*$\}$
$\longrightarrow$ *L-H-M* σ *v* = *L-H-M* ($\sigma \cup \{m\}$) *v*
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix σ *m v*
assume $\sigma \in \Sigma \wedge m \in M \wedge v \in V$ *immediately-next-message* (σ, *m*) $v \in V - \{$*sender m*$\}$
show *L-H-M* σ *v* = *L-H-M* ($\sigma \cup \{m\}$) *v*
proof (*cases* $v \in$ *equivocating-validators* σ)
case *True*
then show *?thesis*
apply (*simp add*: *L-H-M-def*)
using ‹$\sigma \in \Sigma \wedge m \in M \wedge v \in V$› ‹*immediately-next-message* (σ, *m*)› ‹$v \in V - \{$*sender m*$\}$› *equivocation-status-of-non-sender-not-affected-by-minimal-transitions*
by *auto*
next
case *False*
then have $v \notin$ *equivocating-validators* $\sigma \wedge v \notin$ *equivocating-validators* ($\sigma \cup \{m\}$)
using *equivocation-status-of-non-sender-not-affected-by-minimal-transitions* ‹$\sigma \in \Sigma \wedge m \in M \wedge v \in V$› ‹*immediately-next-message* (σ, *m*)› ‹$v \in V - \{$*sender m*$\}$›
by *auto*
then show *?thesis*
apply (*simp add*: *L-H-M-def L-M-def*)

using ‹$\sigma \in \Sigma \wedge m \in M \wedge v \in V$› ‹*immediately-next-message* (σ, m)› ‹$v \in V$ $-$ {*sender m*}›
later-from-of-non-sender-not-affected-by-minimal-transitions
from-sender-of-non-sender-not-affected-by-minimal-transitions
by (*metis* (*no-types*, *lifting*) *Un-insert-right from-sender-type-for-state subsetCE sup-bot.right-neutral*)
qed
qed

lemma (**in** *Protocol*) *agreeing-status-of-non-sender-not-affected-by-minimal-transitions*
:
$\forall$ σ *m v p*. $\sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow$ $v \in V -$ {*sender m*}
$\longrightarrow$ $v \in$ *agreeing-validators* $(p, \sigma) \longleftrightarrow v \in$ *agreeing-validators* $(p, \sigma \cup \{m\})$
apply (*simp add*: *agreeing-validators-def agreeing-def L-H-E-def observed-non-equivocating-validators-def observed-def*)
using *L-H-M-of-non-sender-not-affected-by-minimal-transitions*
equivocation-status-of-non-sender-not-affected-by-minimal-transitions
by *auto*

lemma (**in** *Protocol*) *L-H-J-of-non-sender-not-affected-by-minimal-transitions* :
$\forall$ σ *m v*. $\sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow$ $v \in V -$ {*sender m*}
$\longrightarrow$ *L-H-J* σ v = *L-H-J* $(\sigma \cup \{m\})$ v
apply (*simp add*: *L-H-J-def*)
using *L-H-M-of-non-sender-not-affected-by-minimal-transitions*
by *auto*

lemma (**in** *Protocol*) *later-disagreeing-of-non-sender-not-affected-by-minimal-transitions*
:
$\forall$ σ *m m′ v*. $\sigma \in \Sigma \wedge m \in M \wedge m' \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, m')
$\longrightarrow$ $v \in V -$ {*sender m′*}
$\longrightarrow$ *later-disagreeing-messages* (p, m, v, σ) = *later-disagreeing-messages* $(p, m, v, \sigma \cup \{m'\})$
apply (*simp add*: *later-disagreeing-messages-def*)
using *later-from-of-non-sender-not-affected-by-minimal-transitions* **by** *auto*

lemma (**in** *Protocol*) *inspector-preserved-over-message-from-non-member* :
$\forall$ σ *m v-set p*. $\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow$ *sender m* $\notin$ *v-set*

```
  ⟶ inspector (v-set, σ, p)
  ⟶ inspector (v-set, σ ∪ {m}, p)
  apply (rule, rule, rule, rule, rule, rule, rule, rule)
proof −
  fix σ m v-set p
  assume σ ∈ Σ ∧ m ∈ M ∧ v-set ⊆ V immediately-next-message (σ, m) sender
m ∉ v-set inspector (v-set, σ, p)

  then have ∀ v ∈ v-set. v ∈ agreeing-validators (p, σ) ⟶ v ∈ agreeing-validators
(p, σ ∪ {m})
    using agreeing-status-of-non-sender-not-affected-by-minimal-transitions
    by blast

  moreover have ∀ v ∈ v-set.
                         (∀ v-set'. gt-threshold(v-set', the-elem (L-H-J σ v)) ⟶
gt-threshold(v-set', the-elem (L-H-J (σ ∪ {m}) v)))
   using ‹σ ∈ Σ ∧ m ∈ M ∧ v-set ⊆ V› ‹immediately-next-message (σ, m)› ‹sender
m ∉ v-set›
        L-H-J-of-non-sender-not-affected-by-minimal-transitions
    by fastforce

  moreover have ∀ v ∈ v-set.
                    (∀ v-set'. v-set' ⊆ v-set ∧
                        (∀ v' ∈ v-set'.
                            v' ∈ agreeing-validators (p, (the-elem (L-H-J σ v)))
                          ∧ later-disagreeing-messages (p, the-elem (L-H-M (the-elem
(L-H-J σ v)) v'), v', σ) = ∅)
                    ⟶ (∀ v' ∈ v-set'.
                          v' ∈ agreeing-validators (p, (the-elem (L-H-J (σ ∪ {m}) v)))
                          ∧ later-disagreeing-messages (p, the-elem (L-H-M (the-elem
(L-H-J (σ ∪ {m}) v)) v'), v', (σ ∪ {m})) = ∅))
    apply (rule, rule, rule, rule)
  proof −
    fix v v-set' v'
    assume v ∈ v-set
    and a1: v-set' ⊆ v-set ∧ (∀ v' ∈ v-set'.
        v' ∈ agreeing-validators (p, the-elem (L-H-J σ v)) ∧ later-disagreeing-messages
(p, the-elem (L-H-M (the-elem (L-H-J σ v)) v'), v', σ) = ∅)
    and v' ∈ v-set'
   then have l1: v' ∈ agreeing-validators (p, the-elem (L-H-J σ v)) ∧ later-disagreeing-messages
(p, the-elem (L-H-M (the-elem (L-H-J σ v)) v'), v', σ) = ∅
      by blast
    have v ∈ observed-non-equivocating-validators σ
    using ‹v ∈ v-set› ‹inspector (v-set, σ, p)› inspector-imps-everyone-observed-non-equivocating
         ‹σ ∈ Σ ∧ m ∈ M ∧ v-set ⊆ V› by blast
    have v' ∈ observed-non-equivocating-validators (the-elem (L-H-J σ v))
      using l1 by (simp add: agreeing-validators-def)
    then have v' ∈ V − {sender m}
      using ‹σ ∈ Σ ∧ m ∈ M ∧ v-set ⊆ V› ‹sender m ∉ v-set› ‹v' ∈ v-set'›
```

a1 **by** *blast*
then moreover have *the-elem* (*L-H-J* σ *v*) = *the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)
using *L-H-J-of-non-sender-not-affected-by-minimal-transitions* ‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› ‹*immediately-next-message* (σ, *m*)› ‹*sender m* $\notin$ *v-set*› ‹*v* $\in$ *v-set*›
by (*metis* (*no-types*, *lifting*) *M-type* ‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› *insert-Diff insert-iff subsetCE*)
then moreover have *the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ *v*)) *v'*) = *the-elem* (*L-H-M* (*the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)) *v'*)
using *L-H-M-of-non-sender-not-affected-by-minimal-transitions*
by *simp*
then moreover have *later-disagreeing-messages* (*p*, *the-elem* (*L-H-M* (*the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)) *v'*), *v'*, $\sigma \cup \{m\}$) = $\emptyset$
proof −
have *ll1*: *later-disagreeing-messages* (*p*, *the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ *v*)) *v'*), *v'*, σ) = *later-disagreeing-messages* (*p*, *the-elem* (*L-H-M* (*the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)) *v'*), *v'*, σ)
by (*simp add*: *calculation*(*2*))
have $\sigma \cup \{m\} \in \Sigma \wedge v \in V$
using ‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› ‹*immediately-next-message* (σ, *m*)› *state-transition-only-made-by-immediately-next-message*
‹*v* $\in$ *v-set*› **by** *blast*
hence *the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*) $\in \Sigma$
using *L-H-J-type L-H-J-of-observed-non-equivocating-validator-is-singleton* ‹*v* $\in$ *observed-non-equivocating-validators* σ›
by (*metis* ‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› *calculation*(*2*) *insert-subset is-singleton-the-elem*)
hence *the-elem* (*L-H-M* (*the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)) *v'*) $\in M$
using *L-H-M-type L-H-M-of-observed-non-equivocating-validator-is-singleton* ‹*v'* $\in$ *observed-non-equivocating-validators* (*the-elem* (*L-H-J* σ *v*))›
using *L-H-M-is-in-the-state calculation*(*2*) *state-is-subset-of-M* **by** *fastforce*
hence *later-disagreeing-messages* (*p*, *the-elem* (*L-H-M* (*the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)) *v'*), *v'*, σ) = *later-disagreeing-messages* (*p*, *the-elem* (*L-H-M* (*the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)) *v'*), *v'*, $\sigma \cup \{m\}$)
using *later-disagreeing-of-non-sender-not-affected-by-minimal-transitions ll1*
‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› ‹*immediately-next-message* (σ, *m*)› ‹*v'* $\in V - \{$*sender m*$\}$›
by *auto*
then show *?thesis*
using *l1 ll1* **by** *blast*
qed
ultimately show *v'* $\in$ *agreeing-validators* (*p*, *the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)) $\wedge$
later-disagreeing-messages (*p*, *the-elem* (*L-H-M* (*the-elem* (*L-H-J* ($\sigma \cup \{m\}$) *v*)) *v'*), *v'*, $\sigma \cup \{m\}$) = $\emptyset$
using *later-disagreeing-of-non-sender-not-affected-by-minimal-transitions l1*
‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› ‹*immediately-next-message* (σ, *m*)› ‹*v'* $\in V - \{$*sender m*$\}$›
by *simp*

```
  qed
  ultimately show inspector (v-set, σ ∪ {m}, p)
    using ‹inspector (v-set, σ, p)›
    apply (simp add: inspector-def)
    by meson
qed
```

```
lemma (in Protocol) later-messages-from-non-equivocating-validator-include-all-earlier-messages
:
  ∀ v σ σ1 σ2. σ ∈ Σ ∧ σ1 ∈ Σ ∧ σ1 ⊆ σ ∧ σ2 ⊆ σ ∧ σ1 ∩ σ2 = ∅
  ⟶ (∀ m1 ∈ σ1. sender m1 = v
      ⟶ (∀ m2 ∈ σ2. sender m2 = v ⟶ m1 ∈ justification m2))
  using strict-subset-of-state-have-immediately-next-messages
  apply (simp add: immediately-next-message-def)
  sorry
```

```
lemma (in Protocol) new-message-is-L-H-M-of-sender :
  ∀ σ m v. σ ∈ Σ ∧ m ∈ M
  ⟶ immediately-next-message (σ, m)
  ⟶ sender m ∉ equivocating-validators (σ ∪ {m})
  ⟶ m = the-elem (L-H-M (σ ∪ {m}) (sender m))
  using L-H-M-of-observed-non-equivocating-validator-is-singleton
  sorry
```

```
lemma (in Protocol) new-justification-is-L-H-J-of-sender :
  ∀ σ m v. σ ∈ Σ ∧ m ∈ M
  ⟶ immediately-next-message (σ, m)
  ⟶ sender m ∉ equivocating-validators (σ ∪ {m})
  ⟶ the-elem (L-H-J (σ ∪ {m}) (sender m)) = justification m
  using new-message-is-L-H-M-of-sender
  apply (simp add: L-H-J-def)
  using L-H-M-of-observed-non-equivocating-validator-is-singleton
  sorry
```

lemma (**in** *Protocol*) *L-H-M-of-others-for-sender-is-the-previous-one-or-later*:
  $\forall$ *σ m v*. *σ* $\in \Sigma \wedge$ *m* $\in$ *M* $\wedge$ *v* $\in$ *V*
  $\longrightarrow$ *immediately-next-message* (*σ*, *m*)
  $\longrightarrow$ *sender m* $\notin$ *equivocating-validators* (*σ* $\cup$ {*m*})
  $\longrightarrow$ *v* $\notin$ *equivocating-validators σ*
  $\longrightarrow$ *the-elem* (*L-H-M* (*justification m*) *v*) = *the-elem* (*L-H-M* (*the-elem* (*L-H-J* *σ* (*sender m*))) *v*)
      $\vee$ *justified* (*the-elem* (*L-H-M* (*the-elem* (*L-H-J σ* (*sender m*))) *v*)) (*the-elem* (*L-H-M* (*justification m*) *v*))
  sorry

lemma (**in** *Protocol*) *justified-message-exists-in-later-from*:
  $\forall$ *σ m1 m2*. *σ* $\in \Sigma \wedge$ {*m1*, *m2*} $\subseteq$ *σ*
  $\longrightarrow$ *justified m1 m2*
  $\longrightarrow$ *m2* $\in$ *later-from* (*m1*, *sender m2*, *σ*)
  by (*simp add*: *later-from-def later-def from-sender-def*)

lemma (**in** *Protocol*) *new-message-see-all-members-agreeing* :
  $\forall$ *σ m v-set p*. *σ* $\in \Sigma \wedge$ *m* $\in$ *M* $\wedge$ *v-set* $\subseteq$ *V*
  $\longrightarrow$ *immediately-next-message* (*σ*, *m*)
  $\longrightarrow$ *sender m* $\in$ *v-set*
  $\longrightarrow$ *sender m* $\notin$ *equivocating-validators* (*σ* $\cup$ {*m*})
  $\longrightarrow$ *inspector* (*v-set*, *σ*, *p*)
  $\longrightarrow$ *v-set* $\subseteq$ *agreeing-validators* (*p*, *justification m*)
  sorry

lemma (**in** *Protocol*) *new-message-from-member-see-itself-agreeing* :
  $\forall$ *σ m v-set p*. *σ* $\in \Sigma \wedge$ *m* $\in$ *M* $\wedge$ *v-set* $\subseteq$ *V*
  $\longrightarrow$ *immediately-next-message* (*σ*, *m*)
  $\longrightarrow$ *sender m* $\in$ *v-set*
  $\longrightarrow$ *sender m* $\notin$ *equivocating-validators* (*σ* $\cup$ {*m*})
  $\longrightarrow$ *inspector* (*v-set*, *σ*, *p*)
  $\longrightarrow$ *sender m* $\in$ *agreeing-validators* (*p*, *justification m*)
  using *new-message-see-all-members-agreeing*
  by *blast*

lemma (**in** *Protocol*) *L-H-M-of-sender-is-previous-L-H-M* :

$\forall\ \sigma\ m.\ \sigma \in \Sigma \land m \in M$
$\longrightarrow$ *immediately-next-message* $(\sigma,\ m)$
$\longrightarrow$ *sender m* $\notin$ *equivocating-validators* $(\sigma \cup \{m\})$
$\longrightarrow$ *the-elem* (*L-H-M* (*justification m*) (*sender m*)) = *the-elem* (*L-H-M* σ (*sender m*))
sorry

lemma (**in** *Protocol*) *L-H-M-of-sender-justified-by-new-message* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \land m \in M$
$\longrightarrow$ *immediately-next-message* $(\sigma,\ m)$
$\longrightarrow$ *sender m* $\notin$ *equivocating-validators* $(\sigma \cup \{m\})$
$\longrightarrow$ *justified* (*the-elem* (*L-H-M* σ (*sender m*))) *m*

using *justification-is-total-on-messages-from-non-equivocating-validator*
sorry

lemma (**in** *Protocol*) *nothing-later-than-L-H-M* :
$\forall\ \sigma\ m\ v.\ \sigma \in \Sigma \land m \in M \land v \in V$
$\longrightarrow v \notin$ *equivocating-validators* σ
$\longrightarrow$ *later-from* (*the-elem* (*L-H-M* σ *v*), *v*, σ) = $\emptyset$
apply (*simp add*: *later-from-def L-H-M-def L-M-def from-sender-def justified-def equivocating-validators-def is-equivocating-def*)
sorry

lemma (**in** *Protocol*) *later-messages-for-sender-is-only-new-message* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \land m \in M$
$\longrightarrow$ *immediately-next-message* $(\sigma,\ m)$
$\longrightarrow$ *sender m* $\notin$ *equivocating-validators* $(\sigma \cup \{m\})$
$\longrightarrow$ *later-from* (*the-elem* (*L-H-M* σ (*sender m*)), *sender m*, $\sigma \cup \{m\}$) = $\{m\}$
sorry

lemma (**in** *Protocol*) *later-disagreeing-is-monotonic*:
$\forall\ v\ \sigma$ *m1 m2 p*. $v \in V \land \sigma \in \Sigma \land \{m1,\ m2\} \subseteq M$
$\longrightarrow$ *justified m1 m2*
$\longrightarrow$ *later-disagreeing-messages* (*p*, *m2*, *v*, σ) $\subseteq$ *later-disagreeing-messages* (*p*, *m1*, *v*, σ)
using *message-in-state-is-strict-subset-of-the-state message-in-state-is-valid M-type state-is-in-pow-Mi*
apply (*simp add*: *later-disagreeing-messages-def later-from-def justified-def*)
by *auto*

lemma (**in** *Protocol*) *previous-empty-later-disagreeing-messages-imps-empty-in-new-message* :

∀ σ m v p. σ ∈ Σ ∧ m ∈ M ∧ v ∈ V
⟶ *immediately-next-message* (σ, m)
⟶ *sender m* ∉ *equivocating-validators* (σ ∪ {m})
⟶ *later-disagreeing-messages* (p, (*the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ (*sender* m))) v)), v, σ) = ∅
⟶ *later-disagreeing-messages* (p, (*the-elem* (*L-H-M* (*justification* m) v)), v, σ) = ∅
apply (*simp add*: *later-disagreeing-messages-def*)
sorry

lemma (**in** *Protocol*) *inspector-preserved-over-message-from-non-equivocating-member* :
∀ σ m *v-set* p. σ ∈ Σt ∧ m ∈ M ∧ *v-set* ⊆ V
⟶ *finite v-set*
⟶ *majority-driven p*
⟶ *immediately-next-message* (σ, m)
⟶ *sender m* ∈ *v-set*
⟶ *sender m* ∉ *equivocating-validators* (σ ∪ {m})
⟶ *inspector* (*v-set*, σ, p)
⟶ *inspector* (*v-set*, σ ∪ {m}, p)
apply (*rule*+)
proof −
fix σ m *v-set* p
assume σ ∈ Σt ∧ m ∈ M ∧ *v-set* ⊆ V *finite v-set majority-driven p immediately-next-message* (σ, m) *sender m* ∈ *v-set*
sender m ∉ *equivocating-validators* (σ ∪ {m}) *inspector* (*v-set*, σ, p)

then have σ ∪ {m} ∈ Σt
by (*metis* (*no-types*, *lifting*) Σ*t-def equivocating-validators-preserved-over-honest-message equivocation-fault-weight-def is-faults-lt-threshold-def mem-Collect-eq state-transition-by-immediately-next-mess*

then have *sender m* ∈ *observed-non-equivocating-validators* (σ ∪ {m})
using *inspector-imps-everyone-observed-non-equivocating* ‹*inspector* (*v-set*, σ, p)› ‹σ ∈ Σt ∧ m ∈ M ∧ *v-set* ⊆ V› ‹*sender m* ∉ *equivocating-validators* (σ ∪ {m})›
apply (*simp add*: *observed-non-equivocating-validators-def observed-def*)
by *blast*
then have *the-elem* (*L-H-J* (σ ∪ {m}) (*sender m*)) = *justification m*
using *new-justification-is-L-H-J-of-sender*
‹σ ∈ Σt ∧ m ∈ M ∧ *v-set* ⊆ V› ‹*immediately-next-message* (σ, m)› ‹*sender m* ∉ *equivocating-validators* (σ ∪ {m})›
by (*simp add*: Σ*t-def*)

then moreover have ∀ v ∈ *v-set*.
(∀ *v-set*′. *v-set*′ ⊆ *v-set* ∧ *gt-threshold*(*v-set*′, *the-elem* (*L-H-J* σ v)) ⟶ *gt-threshold*(*v-set*′, *the-elem* (*L-H-J* (σ ∪ {m}) v)))
using ‹σ ∈ Σt ∧ m ∈ M ∧ *v-set* ⊆ V› ‹*immediately-next-message* (σ, m)›

‹*sender m* ∈ *v-set*›
L-H-J-of-non-sender-not-affected-by-minimal-transitions
sorry

then moreover have ∀ *v* ∈ *v-set*. *v* ∈ *agreeing-validators* (*p*, σ ∪ {*m*})
proof −
have *sender m* ∈ *agreeing-validators* (*p*, σ ∪ {*m*})
proof −
have ∀ *v-set*′. *v-set*′ ⊆ *v-set* ⟶ *v-set*′ ⊆ *agreeing-validators* (*p*, *the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*)))
using *new-message-see-all-members-agreeing*
by (*smt Protocol.new-message-see-all-members-agreeing Protocol-axioms* Σ*t-is-subset-of-*Σ ‹σ ∈ Σ*t* ∧ *m* ∈ *M* ∧ *v-set* ⊆ *V*› ‹*immediately-next-message* (σ, *m*)› ‹*inspector* (*v-set*, σ, *p*)› ‹*sender m* ∈ *v-set*› ‹*sender m* ∉ *equivocating-validators* (σ ∪ {*m*})› ‹*the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*)) = *justification m*› *subsetCE subset-trans*)
have ∃ *v-set*′. *v-set*′ ⊆ *v-set* ∧ *gt-threshold*(*v-set*′, *the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*)))
using ‹*inspector* (*v-set*, σ, *p*)›
apply (*simp add*: *inspector-def*)
using ‹∀ *v*∈*v-set*. ∀ *v-set*′. *v-set*′ ⊆ *v-set* ∧ *gt-threshold* (*v-set*′, *the-elem* (*L-H-J* σ *v*)) ⟶ *gt-threshold* (*v-set*′, *the-elem* (*L-H-J* (σ ∪ {*m*}) *v*))›
‹*sender m* ∈ *v-set*› ‹*the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*)) = *justification m*›
by (*smt Un-insert-right* Σ*t-is-subset-of-*Σ ‹σ ∈ Σ*t* ∧ *m* ∈ *M* ∧ *v-set* ⊆ *V*› ‹*immediately-next-message* (σ, *m*)› ‹*inspector* (*v-set*, σ, *p*)› ‹*sender m* ∉ *equivocating-validators* (σ ∪ {*m*})› *subsetCE subset-trans sup-bot.right-neutral*)
then have ∃ *v-set*′. *v-set*′ ⊆ *V* ∧ *finite v-set*′
∧ *v-set*′ ⊆ *agreeing-validators* (*p*, *the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*))) ∧ *gt-threshold*(*v-set*′, *the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*)))
using ‹*finite v-set*› ‹σ ∈ Σ*t* ∧ *m* ∈ *M* ∧ *v-set* ⊆ *V*› ‹∀ *v-set*′. *v-set*′ ⊆ *v-set* ⟶ *v-set*′ ⊆ *agreeing-validators* (*p*, *the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*)))›
by (*meson rev-finite-subset subset-trans*)
then have ∀ *c* ∈ ε (*the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*))). *p c*
using ‹*majority-driven p*› ‹*sender m* ∈ *v-set*› *gt-threshold-imps-estimator-agreeing* ‹σ ∈ Σ*t* ∧ *m* ∈ *M* ∧ *v-set* ⊆ *V*›
‹*sender m* ∈ *observed-non-equivocating-validators* (σ ∪ {*m*})› ‹σ ∪ {*m*} ∈ Σ*t*› ‹*the-elem* (*L-H-J* (σ ∪ {*m*}) (*sender m*)) = *justification m*›
*past-state-of-*Σ*t-is-*Σ*t state-transition-is-immediately-next-message M-type*
unfolding Σ*t-def*
by (*smt* Σ*t-def* Σ*t-is-subset-of-*Σ *is-future-state.simps subsetD*)
then have ∀ *c* ∈ *L-H-E* (σ ∪ {*m*}) (*sender m*). *p c*
using ‹*sender m* ∈ *observed-non-equivocating-validators* (σ ∪ {*m*})› ‹σ ∪ {*m*} ∈ Σ*t*› *L-H-M-of-observed-non-equivocating-validator-is-singleton*
apply (*simp add*: *L-H-E-def L-H-J-def*)
sorry
then show *?thesis*
using ‹*sender m* ∈ *observed-non-equivocating-validators* (σ ∪ {*m*})›
by (*simp add*: *agreeing-validators-def agreeing-def*)

```
    qed
    then show ?thesis
      using agreeing-status-of-non-sender-not-affected-by-minimal-transitions
    by (smt Diff-iff Σt-is-subset-of-Σ ‹σ ∈ Σt ∧ m ∈ M ∧ v-set ⊆ V› ‹immediately-next-message
(σ, m)› ‹inspector (v-set, σ, p)› contra-subsetD empty-iff insert-iff inspector-imps-everyone-agreeing)
  qed

  moreover have ∀ v ∈ v-set.
                  (∀ v-set'. v-set' ⊆ v-set ∧
                      (∀ v' ∈ v-set'.
                          v' ∈ agreeing-validators (p, (the-elem (L-H-J σ v)))
                        ∧ later-disagreeing-messages (p, the-elem (L-H-M (the-elem
(L-H-J σ v)) v'), v', σ) = ∅)
                  ⟶ (∀ v' ∈ v-set'.
                        v' ∈ agreeing-validators (p, (the-elem (L-H-J (σ ∪ {m}) v)))
                        ∧ later-disagreeing-messages (p, the-elem (L-H-M (the-elem
(L-H-J (σ ∪ {m}) v)) v'), v', (σ ∪ {m})) = ∅))
    apply (rule, rule, rule, rule)
  proof –
    fix v v-set' v'
    assume v ∈ v-set
    and a1: v-set' ⊆ v-set ∧ (∀ v' ∈ v-set'.
        v' ∈ agreeing-validators (p, the-elem (L-H-J σ v)) ∧ later-disagreeing-messages
(p, the-elem (L-H-M (the-elem (L-H-J σ v)) v'), v', σ) = ∅)
    and v' ∈ v-set'
    show v' ∈ agreeing-validators (p, the-elem (L-H-J (σ ∪ {m}) v)) ∧
            later-disagreeing-messages (p, the-elem (L-H-M (the-elem (L-H-J (σ ∪
{m}) v)) v'), v', σ ∪ {m}) = ∅
      sorry
    qed
  ultimately show inspector (v-set, σ ∪ {m}, p)
    using ‹inspector (v-set, σ, p)›
    apply (simp add: inspector-def)
    by meson
qed
```

```
lemma (in Protocol) inspector-preserved-over-message-from-equivocating-member
:
  ∀ σ m v-set p. σ ∈ Σ ∧ m ∈ M ∧ v-set ⊆ V
  ⟶ majority-driven p
  ⟶ immediately-next-message (σ, m)
  ⟶ sender m ∈ v-set
  ⟶ sender m ∈ equivocating-validators (σ ∪ {m})
  ⟶ $\sigma \cup \{m\} \in \Sigma t$
  ⟶ inspector (v-set, $\sigma$, p)
  ⟶ inspector (v-set, $\sigma \cup \{m\}$, p)

  sorry



lemma (in Protocol) inspector-preserved-over-immediately-next-message :
  ∀ $\sigma$ m v-set p. $\sigma \in \Sigma t$ ∧ v-set ⊆ V
  ⟶ majority-driven p
  ⟶ immediately-next-message ($\sigma$, m)
  ⟶ $\sigma \cup \{m\} \in \Sigma t$
  ⟶ inspector (v-set, $\sigma$, p)
  ⟶ inspector (v-set, $\sigma \cup \{m\}$, p)
  using inspector-preserved-over-message-from-non-member
       inspector-preserved-over-message-from-non-equivocating-member
       inspector-preserved-over-message-from-equivocating-member
  apply (simp add: Σt-def)
  by (metis V-type insert-iff message-in-state-is-valid rev-finite-subset)



lemma (in Protocol) inspector-preserved-in-future:
  ∀ $\sigma$ v-set p. $\sigma \in \Sigma t$ ∧ v-set ⊆ V
  ⟶ majority-driven p
  ⟶ inspector (v-set, $\sigma$, p)
  ⟶ (∀ $\sigma'$ ∈ futures $\sigma$. inspector (v-set, $\sigma'$, p))
proof auto
  fix $\sigma$ v-set p $\sigma'$
  assume $\sigma \in \Sigma t$ v-set ⊆ V majority-driven p inspector (v-set, $\sigma$, p) $\sigma'$ ∈ futures
$\sigma$
  hence $\sigma \in \Sigma$ $\sigma' \in \Sigma$
    unfolding Σt-def futures-def
    by auto

  have $\sigma' \in \Sigma t$
    using ‹$\sigma'$ ∈ futures $\sigma$› futures-def by blast

  obtain message-list where MessagePath $\sigma$ $\sigma'$ message-list
    using ‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$› ‹$\sigma'$ ∈ futures $\sigma$› exist-message-path by blast

  have ⟦ MessagePath $\sigma$ $\sigma'$ message-list; $\sigma \in \Sigma$; $\sigma' \in \Sigma$; $\sigma' \in \Sigma t$; inspector (v-set,
$\sigma$, p) ⟧ ⟹ inspector (v-set, $\sigma'$, p)
    apply (induct length message-list arbitrary: message-list $\sigma$ $\sigma'$)
    apply (simp)
    using coherent-nil-message-path apply auto[1]
  proof−
    fix n message-list $\sigma$ $\sigma'$
```

assume $\bigwedge$*message-list* σ σ'.
n = *length message-list* $\Longrightarrow$ *MessagePath* σ σ' *message-list* $\Longrightarrow$ $\sigma \in \Sigma$ $\Longrightarrow$ $\sigma' \in \Sigma$ $\Longrightarrow$ $\sigma' \in \Sigma t$ $\Longrightarrow$ *inspector* (*v-set*, σ, *p*) $\Longrightarrow$ *inspector* (*v-set*, σ', *p*)
and *Suc n* = *length message-list MessagePath* σ σ' *message-list* $\sigma \in \Sigma$ $\sigma' \in \Sigma$ $\sigma' \in \Sigma t$ *inspector* (*v-set*, σ, *p*)

obtain *m ms* **where** *message-list* = *m* # *ms MessagePath* ($\sigma \cup \{m\}$) σ' *ms immediately-next-message* (σ,*m*) $\sigma \cup \{m\} \in \Sigma$
by (*metis* ‹*MessagePath* σ σ' *message-list*› ‹*Suc n* = *length message-list*› *coherent-nonnil-message-path nat.distinct*(*1*))
hence $\sigma \in \Sigma t$ $\sigma \cup \{m\} \in \Sigma t$
apply (*meson* ‹*MessagePath* σ σ' *message-list*› ‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma t$› *coherent-message-path-inclusive is-future-state.simps past-state-of-*Σ*t-is-*Σ*t*)
apply (*meson* ‹*MessagePath* ($\sigma \cup \{m\}$) σ' *ms*› ‹$\sigma \cup \{m\} \in \Sigma$› ‹$\sigma' \in \Sigma t$› *coherent-message-path-inclusive is-future-state.simps past-state-of-*Σ*t-is-*Σ*t*)
done
have *inspector* (*v-set*, $\sigma \cup \{m\}$, *p*)
using ‹$\sigma \in \Sigma t$› ‹$\sigma \cup \{m\} \in \Sigma t$› ‹*immediately-next-message* (σ, *m*)› ‹*inspector* (*v-set*, σ, *p*)› ‹*majority-driven p*› ‹*v-set* $\subseteq$ *V*› *inspector-preserved-over-immediately-next-message* **by** *blast*
moreover have *n* = *length ms*
using ‹*Suc n* = *length message-list*› ‹*message-list* = *m* # *ms*› **by** *auto*
moreover have *MessagePath* ($\sigma \cup \{m\}$) σ' *ms*
using ‹*MessagePath* ($\sigma \cup \{m\}$) σ' *ms*› **by** *auto*
moreover have $\sigma \cup \{m\} \in \Sigma$ $\sigma' \in \Sigma$
using ‹$\sigma \cup \{m\} \in \Sigma$› **apply** *auto*[*1*]
by (*simp add*: ‹$\sigma' \in \Sigma$›)
moreover have $\sigma \cup \{m\} \in \Sigma t$
using ‹$\sigma \cup \{m\} \in \Sigma t$› **by** *auto*
ultimately show *inspector* (*v-set*, σ', *p*)
using ‹$\bigwedge$*message-list* σ' σ. $[\![$*n* = *length message-list*; *MessagePath* σ σ' *message-list*; $\sigma \in \Sigma$; $\sigma' \in \Sigma$; $\sigma' \in \Sigma t$; *inspector* (*v-set*, σ, *p*)$]\!]$ $\Longrightarrow$ *inspector* (*v-set*, σ', *p*)› ‹$\sigma' \in \Sigma t$› **by** *blast*
qed

thus *inspector* (*v-set*, σ', *p*)
using ‹*MessagePath* σ σ' *message-list*› ‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$› ‹$\sigma' \in \Sigma t$› ‹*inspector* (*v-set*, σ, *p*)› **by** *blast*
qed

lemma (**in** *Protocol*) *inspector-is-safety-oracle* :
$\forall$ σ *v-set p*. $\sigma \in \Sigma t$ $\wedge$ *v-set* $\subseteq$ *V*
$\longrightarrow$ *finite v-set*
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *inspector* (*v-set*, σ, *p*)
$\longrightarrow$ *state-property-is-decided* (*naturally-corresponding-state-property p*, σ)
using *inspector-preserved-in-future inspector-imps-estimator-agreeing*

```
 apply (simp add: naturally-corresponding-state-property-def futures-def state-property-is-decided-def)
 by meson

end
theory TFGCasper

imports Main HOL.Real CBCCasper LatestMessage CliqueOracle ConsensusSafety

begin
```

```
locale BlockchainParams = Params +
  fixes genesis :: consensus-value

  and prev :: consensus-value ⇒ consensus-value


fun (in BlockchainParams) n-cestor :: consensus-value ∗ nat ⇒ consensus-value
  where
    n-cestor (b, 0) = b
  | n-cestor (b, n) = n-cestor (prev b, n−1)


definition (in BlockchainParams) blockchain-membership :: consensus-value ⇒
consensus-value ⇒ bool (infixl ⇂ 70)
  where
    b1 ⇂ b2 = (∃ n. n ∈ ℕ ∧ b1 = n-cestor (b2, n))

notation (ASCII)
  comp  (infixl blockchain-membership 70)

lemma (in BlockchainParams) prev-membership :
  prev b ⇂ b
  apply (simp add: blockchain-membership-def)
 by (metis BlockchainParams.n-cestor.simps(1) BlockchainParams.n-cestor.simps(2)
Nats-1 One-nat-def diff-Suc-1)

definition (in BlockchainParams) block-conflicting :: (consensus-value ∗ consensus-value)
⇒ bool
  where
    block-conflicting = (λ(b1, b2). ¬ (b1 ⇂ b2 ∨ b2 ⇂ b1))

lemma (in BlockchainParams) n-cestor-transitive :
  ∀ n1 n2 x y z. {n1, n2} ⊆ ℕ
    ⟶ x = n-cestor (y, n1)
```

    ⟶ *y* = *n-cestor* (*z*, *n2*)
    ⟶ *x* = *n-cestor* (*z*, *n1* + *n2*)
  apply (*rule*, *rule*)
proof −
  fix *n1 n2*
  show ∀ *x y z*. {*n1*, *n2*} ⊆ ℕ ⟶ *x* = *n-cestor* (*y*, *n1*) ⟶ *y* = *n-cestor* (*z*, *n2*)
⟶ *x* = *n-cestor* (*z*, *n1* + *n2*)
    apply (*induction n2*)
    apply *simp*
    apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
  proof −
    fix *n2 x y z*
    assume ∀ *x y z*. {*n1*, *n2*} ⊆ ℕ ⟶ *x* = *n-cestor* (*y*, *n1*) ⟶ *y* = *n-cestor* (*z*,
n2) ⟶ *x* = *n-cestor* (*z*, *n1* + *n2*)
    assume {*n1*, *Suc n2*} ⊆ ℕ
    assume *x* = *n-cestor* (*y*, *n1*)
    assume *y* = *n-cestor* (*z*, *Suc n2*)
    then have *y* = *n-cestor* (*prev z*, *n2*)
      by *simp*
    have {*n1*, *n2*} ⊆ ℕ
      by (*simp add*: *Nats-def*)
    then have *x* = *n-cestor* (*prev z*, *n1* + *n2*)
      using ‹*x* = *n-cestor* (*y*, *n1*)› ‹*y* = *n-cestor* (*prev z*, *n2*)›
            ‹∀ *x y z*. {*n1*, *n2*} ⊆ ℕ ⟶ *x* = *n-cestor* (*y*, *n1*) ⟶ *y* = *n-cestor* (*z*,
n2) ⟶ *x* = *n-cestor* (*z*, *n1* + *n2*)›
      by *simp*
    then show *x* = *n-cestor* (*z*, *n1* + *Suc n2*)
      by *simp*
  qed
qed

lemma (**in** *BlockchainParams*) *transitivity-of-blockchain-membership* :
  b1 ⇂ *b2* ∧ *b2* ⇂ *b3* ⟹ *b1* ⇂ *b3*
  apply (*simp add*: *blockchain-membership-def*)
  using *n-cestor-transitive*
  by (*metis id-apply of-nat-eq-id of-nat-in-Nats subsetI*)

lemma (**in** *BlockchainParams*) *irreflexivity-of-blockchain-membership* :
  b ⇂ *b*
  apply (*simp add*: *blockchain-membership-def*)
  using *Nats-0* **by** *fastforce*

definition (**in** *BlockchainParams*) *block-membership* :: *consensus-value* ⇒ *consensus-value-property*
  where
    block-membership b = (λ*b*′. *b* ⇂ *b*′)

lemma (**in** *BlockchainParams*) *also-agreeing-on-ancestors* :

$b' \downharpoonright b \Longrightarrow$ *agreeing* (*block-membership* b, σ, v) $\Longrightarrow$ *agreeing* (*block-membership* b', σ, v)
apply (*simp add*: *agreeing-def block-membership-def*)
using *BlockchainParams.transitivity-of-blockchain-membership* **by** *blast*

definition (**in** *BlockchainParams*) *children* :: *consensus-value* $*$ *state* $\Rightarrow$ *consensus-value set*
where
children $= (\lambda(b, \sigma).\ \{b' \in$ *est* ‘$\sigma.\ b =$ *prev* $b'\})$

lemma (**in** *BlockchainParams*) *observed-block-is-children-of-prev-block* :
$\forall\ b \in$ *est* ‘$\sigma.\ b \in$ *children* (*prev* b, σ)
by (*simp add*: *children-def*)

lemma (**in** *BlockchainParams*) *children-membership* :
$\forall\ b \in$ *children* $(b', \sigma).\ \ b' \downharpoonright b$
apply (*simp add*: *children-def*)
by (*metis BlockchainParams.blockchain-membership-def BlockchainParams.n-cestor.simps*(*2*) *diff-Suc-1 id-apply n-cestor.simps*(*1*) *of-nat-eq-id of-nat-in-Nats*)

locale *Blockchain* = *BlockchainParams* + *Protocol* +

assumes *blockchain-type* : $\forall\ b\ b'\ b''.\ \{b, b', b''\} \subseteq C \longrightarrow b' \downharpoonright b \wedge b'' \downharpoonright b \longrightarrow (b' \downharpoonright b'' \vee b'' \downharpoonright b')$
and *children-conflicting* : $\forall\ \sigma \in \Sigma.\ \forall\ b\ b1\ b2.\ \{b, b1, b2\} \subseteq C \wedge \{b1, b2\} \subseteq$ *children* $(b, \sigma) \longrightarrow$ *block-conflicting* $(b1, b2)$
and *prev-type* : $\forall\ b.\ b \in C \longleftrightarrow$ *prev* $b \in C$
and *genesis-type* : *genesis* $\in C\ \forall\ b \in C.$ *genesis* $\downharpoonright b$ *prev genesis* = *genesis*

lemma (**in** *Blockchain*) *children-type* :
$\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow$ *children* $(b, \sigma) \subseteq C$
apply (*simp add*: *children-def*)
using *prev-type* **by** *auto*

lemma (**in** *Blockchain*) *children-finite* :
$\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow$ *finite* (*children* (b, σ))
apply (*simp add*: *children-def*)
using *state-is-finite*
by *simp*

lemma (**in** *Blockchain*) *conflicting-blocks-imps-conflicting-decision* :
$\forall\ b1\ b2\ \sigma.\ \{b1, b2\} \subseteq C \wedge \sigma \in \Sigma$

```
    ⟶ block-conflicting (b1, b2)
    ⟶ consensus-value-property-is-decided (block-membership b1, σ)
   ⟶ consensus-value-property-is-decided (consensus-value-property-not (block-membership
b2), σ)
  apply (simp add: block-membership-def consensus-value-property-is-decided-def
          naturally-corresponding-state-property-def  state-property-is-decided-def)
  apply (rule, rule, rule, rule, rule, rule)
proof −
  fix b1 b2 σ
 assume b1 ∈ C ∧ b2 ∈ C ∧ σ ∈ Σ and block-conflicting (b1, b2) and ∀ σ∈futures
σ. ∀ b'∈ε σ. b1 ⇂ b'
  show  ∀ σ∈futures σ. ∀ c∈ε σ. ¬ b2 ⇂ c
  proof (rule ccontr)
    assume ¬ (∀ σ∈futures σ. ∀ c∈ε σ. ¬ b2 ⇂ c)
    hence ∃ σ ∈futures σ. ∃ c ∈ ε σ. b2 ⇂ c
      by blast
    hence ∃ σ ∈futures σ. ∃ c ∈ ε σ. b2 ⇂ c ∧ b1 ⇂ c
      using ‹∀ σ∈futures σ. ∀ b'∈ε σ. b1 ⇂ b'› by simp
    hence b1 ⇂ b2 ∨ b2 ⇂ b1
      using blockchain-type
      apply (simp)
     using Σt-is-subset-of-Σ ‹b1 ∈ C ∧ b2 ∈ C ∧ σ ∈ Σ› estimates-are-subset-of-C
futures-def by blast
    then show False
      using ‹block-conflicting (b1, b2)›
      by (simp add: block-conflicting-def)
  qed
qed

theorem (in Blockchain) blockchain-safety :
  ∀ σ-set. σ-set ⊆ Σt
  ⟶ finite σ-set
  ⟶ is-faults-lt-threshold (⋃ σ-set)
  ⟶ (∀ σ σ' b1 b2. {σ, σ'} ⊆ σ-set ∧ {b1, b2} ⊆ C ∧ block-conflicting (b1, b2)
∧ block-membership b1 ∈ consensus-value-property-decisions σ
      ⟶ block-membership b2 ∉ consensus-value-property-decisions σ')
  apply (rule, rule, rule, rule, rule, rule, rule, rule, rule, rule)
proof −
  fix σ-set σ σ' b1 b2
   assume σ-set ⊆ Σt and finite σ-set and is-faults-lt-threshold (⋃σ-set)
  and {σ, σ'} ⊆ σ-set ∧ {b1, b2} ⊆ C ∧ block-conflicting (b1, b2) ∧ block-membership
b1 ∈ consensus-value-property-decisions σ
   and block-membership b2 ∈ consensus-value-property-decisions σ'
  hence ¬ consensus-value-property-is-decided (consensus-value-property-not (block-membership
b1), σ')
       using negation-is-not-decided-by-other-validator ‹σ-set ⊆ Σt› ‹finite σ-set›
‹is-faults-lt-threshold (⋃σ-set)› apply (simp add: consensus-value-property-decisions-def)

        using ‹{σ, σ'} ⊆ σ-set ∧ {b1, b2} ⊆ C ∧ block-conflicting (b1, b2) ∧
```

```
block-membership b1 ∈ consensus-value-property-decisions σ› by auto
   have {b1, b2} ⊆ C ∧ σ ∈ Σ ∧ block-conflicting (b1, b2)
      using Σt-is-subset-of-Σ ‹σ-set ⊆ Σt› ‹{σ, σ'} ⊆ σ-set ∧ {b1, b2} ⊆ C ∧
block-conflicting (b1, b2) ∧ block-membership b1 ∈ consensus-value-property-decisions
σ› by auto
  hence consensus-value-property-is-decided (consensus-value-property-not (block-membership
b1), σ')
    using ‹block-membership b2 ∈ consensus-value-property-decisions σ'› conflicting-blocks-imps-conflicting-dec
     apply (simp add: consensus-value-property-decisions-def)
       by (metis ‹σ-set ⊆ Σt› ‹finite σ-set› ‹is-faults-lt-threshold (⋃σ-set)› ‹{σ,
σ'} ⊆ σ-set ∧ {b1, b2} ⊆ C ∧ block-conflicting (b1, b2) ∧ block-membership b1
∈ consensus-value-property-decisions σ› conflicting-blocks-imps-conflicting-decision
consensus-value-property-decisions-def insert-subset mem-Collect-eq negation-is-not-decided-by-other-validator)

   then show False
       using ‹¬ consensus-value-property-is-decided (consensus-value-property-not
(block-membership b1), σ')› by blast
 qed


theorem (in Blockchain) no-decision-on-conflicting-blocks :
  ∀ σ1 σ2. {σ1, σ2} ⊆ Σt
  ⟶ is-faults-lt-threshold (σ1 ∪ σ2)
  ⟶ (∀ b1 b2. {b1, b2} ⊆ C ∧ block-conflicting (b1, b2)
      ⟶ block-membership b1 ∈ consensus-value-property-decisions σ1
      ⟶ block-membership b2 ∉ consensus-value-property-decisions σ2)
  apply (rule, rule, rule, rule, rule, rule, rule, rule, rule)
proof –
  fix σ1 σ2 b1 b2
  assume {σ1, σ2} ⊆ Σt and is-faults-lt-threshold (σ1 ∪ σ2) and {b1, b2} ⊆ C
∧ block-conflicting (b1, b2)
  and block-membership b1 ∈ consensus-value-property-decisions σ1
  and block-membership b2 ∈ consensus-value-property-decisions σ2
  hence consensus-value-property-is-decided (block-membership b1, σ1)
    by (simp add: consensus-value-property-decisions-def)
 hence ¬ consensus-value-property-is-decided (consensus-value-property-not (block-membership
b1), σ2)
   using two-party-consensus-safety-for-consensus-value-property ‹is-faults-lt-threshold
(σ1 ∪ σ2)› ‹{σ1, σ2} ⊆ Σt› by blast
  have block-membership b2 ∈ consensus-value-property-decisions σ2
    using ‹block-membership b2 ∈ consensus-value-property-decisions σ2›
    by (simp add: consensus-value-property-decisions-def)
  have σ2 ∈ Σt ∧ {b2, b1} ⊆ C ∧ block-conflicting (b2, b1)
    using ‹{σ1, σ2} ⊆ Σt› ‹{b1, b2} ⊆ C ∧ block-conflicting (b1, b2)› by (simp
add: block-conflicting-def)
 hence consensus-value-property-is-decided (consensus-value-property-not (block-membership
b1), σ2)
   using conflicting-blocks-imps-conflicting-decision ‹block-membership b2 ∈ consensus-value-property-decision
σ2›
```

  using *Σt-is-subset-of-Σ consensus-value-property-decisions-def* **by** *auto*
 then show *False*
   using ⟨¬ *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership b1*), *σ2*)⟩ **by** *blast*
qed

definition (**in** *BlockchainParams*) *score* :: *state* ⇒ *consensus-value* ⇒ *real*
 where
  score σ b = *weight-measure* (*agreeing-validators* (*block-membership b*, *σ*))

lemma (**in** *Blockchain*) *unfolding-agreeing-on-block-membership* :
 ∀ *σ* ∈ Σ. *agreeing-validators* (*block-membership b*, *σ*) = {*v* ∈ *V*. ∃ *b′* ∈ *L-H-E σ v*. *b* ⇂ *b′*}
proof −
 have ∀ *v σ*. *v* ∈ *V* ∧ *σ* ∈ Σ ⟶ *v* ∉ *equivocating-validators σ*
   ⟶ (*v* ∈ *observed σ* ∧ (∀ *x* ∈ *L-M σ v*. *b* ⇂ *est x*)) = (*v* ∈ *observed σ* ∧ (∃ *x* ∈*L-M σ v*. *b* ⇂ *est x*))
  using *observed-non-equivocating-validators-have-one-latest-message*
  unfolding *observed-non-equivocating-validators-def is-singleton-def*
  by (*metis Diff-iff empty-iff insert-iff*)
 moreover have ∀ *v σ*. *v* ∈ *V* ∧ *σ* ∈ Σ ⟶ *v* ∉ *equivocating-validators σ*
   ⟶ (*v* ∈ *V* ∧ (∃ *x* ∈*L-M σ v*. *b* ⇂ *est x*)) = (*v* ∈ *observed σ* ∧ (∃ *x* ∈*L-M σ v*. *b* ⇂ *est x*))
  apply (*simp add*: *observed-def L-M-def from-sender-def*)
  by *auto*
 ultimately have ∀ *v σ*. *v* ∈ *V* ∧ *σ* ∈ Σ ⟶ *v* ∉ *equivocating-validators σ*
   ⟶ (*v* ∈ *V* ∧ (∃ *x* ∈*L-M σ v*. *b* ⇂ *est x*)) = (*v* ∈ *observed σ* ∧ (∀ *x* ∈ *L-M σ v*. *b* ⇂ *est x*))
  by *blast*
 then have ∀ *v σ*. *v* ∈ *V* ∧ *σ* ∈ Σ
   ⟶ (*v* ∉ *equivocating-validators σ* ⟶ *v* ∈ *V* ∧ (∃ *x* ∈*L-M σ v*. *b* ⇂ *est x*)) = (*v* ∉ *equivocating-validators σ* ⟶ *v* ∈ *observed σ* ∧ (∀ *x* ∈ *L-M σ v*. *b* ⇂ *est x*))
  by *blast*
 show *?thesis*
  apply (*simp add*: *agreeing-validators-def agreeing-def observed-non-equivocating-validators-def L-H-E-def L-H-M-def block-membership-def*)
  using ⟨∀ *v σ*. *v* ∈ *V* ∧ *σ* ∈ Σ
   ⟶ (*v* ∉ *equivocating-validators σ* ⟶ *v* ∈ *V* ∧ (∃ *x* ∈*L-M σ v*. *b* ⇂ *est x*)) = (*v* ∉ *equivocating-validators σ* ⟶ *v* ∈ *observed σ* ∧ (∀ *x* ∈ *L-M σ v*. *b* ⇂ *est x*))⟩
  observed-type-for-state
  by *blast*

qed

definition (**in** *BlockchainParams*) *score-magnitude* :: *state* ⇒ *consensus-value rel*
  where
    score-magnitude σ = {(*b1*, *b2*). {*b1*, *b2*} ⊆ *C* ∧ *score* σ *b1* ≤ *score* σ *b2*}

lemma (**in** *Blockchain*) *transitivity-of-score-magnitude* :
  ∀ σ ∈ Σ. *trans* (*score-magnitude* σ)
  by (*simp add*: *trans-def score-magnitude-def*)

lemma (**in** *Blockchain*) *reflexivity-of-score-magnitude* :
  ∀ σ ∈ Σ. *refl-on C* (*score-magnitude* σ)
  apply (*simp add*: *refl-on-def score-magnitude-def*)
  by *auto*

lemma (**in** *Blockchain*) *score-magnitude-is-preorder* :
  ∀ σ ∈ Σ. *preorder-on C* (*score-magnitude* σ)
  unfolding *preorder-on-def*
  using *reflexivity-of-score-magnitude transitivity-of-score-magnitude* **by** *simp*

lemma (**in** *Blockchain*) *totality-of-score-magnitude* :
  ∀ σ ∈ Σ. *Relation.total-on C* (*score-magnitude* σ)
  apply (*simp add*: *Relation.total-on-def score-magnitude-def*)
  by *auto*

definition (**in** *BlockchainParams*) *score-magnitude-children* :: *state* ⇒ *consensus-value* ⇒ *consensus-value rel*
  where
    score-magnitude-children σ *b* = {(*b1*, *b2*). {*b1*, *b2*} ⊆ *children* (*b*, σ) ∧ *score* σ *b1* ≤ *score* σ *b2*}

lemma (**in** *Blockchain*) *transitivity-of-score-magnitude-children* :
  ∀ σ ∈ Σ. ∀ *b* ∈ *C*. *trans* (*score-magnitude-children* σ *b*)
  by (*simp add*: *trans-def score-magnitude-children-def*)

lemma (**in** *Blockchain*) *reflexivity-of-score-magnitude-children* :
  ∀ σ ∈ Σ. ∀ *b* ∈ *C*. *refl-on* (*children* (*b*, σ)) (*score-magnitude-children* σ *b*)
  apply (*simp add*: *refl-on-def score-magnitude-children-def*)
  by *blast*

lemma (**in** *Blockchain*) *score-magnitude-children-is-preorder* :
  ∀ σ ∈ Σ. ∀ *b* ∈ *C*. *preorder-on* (*children* (*b*, σ)) (*score-magnitude-children* σ *b*)
  unfolding *preorder-on-def*
  using *reflexivity-of-score-magnitude-children transitivity-of-score-magnitude-children* **by** *simp*

lemma (**in** *Blockchain*) *totality-of-score-magnitude-children* :
  ∀ σ ∈ Σ. ∀ *b* ∈ *C*. *Relation.total-on* (*children* (*b*, σ)) (*score-magnitude-children*

σ b)
  apply (*simp add*: *Relation.total-on-def score-magnitude-children-def*)
  by *auto*

definition (**in** *BlockchainParams*) *best-children* :: *consensus-value* $*$ *state* $\Rightarrow$ *consensus-value set*
  where
    best-children = (λ (b, σ). $\{b' \in C$. *is-arg-max* (*score* σ) ($\lambda b'$. $b' \in$ *children* (b, σ)) $b'\}$)

lemma (**in** *Blockchain*) *best-children-type* :
  $\forall$ b σ. $b \in C \wedge \sigma \in \Sigma \longrightarrow$ *best-children* (b, σ) $\subseteq C$
  by (*simp add*: *is-arg-max-def best-children-def*)

lemma (**in** *Blockchain*) *best-children-finite* :
  $\forall$ b σ. $b \in C \wedge \sigma \in \Sigma \longrightarrow$ *finite* (*best-children* (b, σ))
  apply (*simp add*: *best-children-def is-arg-max-def*)
  using *children-finite*
  by *auto*

lemma (**in** *Blockchain*) *best-children-existence* :
  $\forall$ b σ. $b \in C \wedge \sigma \in \Sigma \longrightarrow$ *children* (b, σ) $\neq \emptyset \longrightarrow$ *best-children* (b, σ) $\in$ *Pow* $C - \{\emptyset\}$
proof −
  have $\forall$ b σ. $b \in C \wedge \sigma \in \Sigma \longrightarrow$ *children* (b, σ) $\neq \emptyset$
    $\longrightarrow$ ($\exists$ b'. *maximum-on-non-strict* (*children* (b, σ)) (*score-magnitude-children* σ b) b')
    using *totality-of-score-magnitude-children score-magnitude-children-is-preorder*
      children-finite children-type connex-preorder-on-finite-non-empty-set-has-maximum
    by *blast*
  then show *?thesis*
   apply (*simp add*: *score-magnitude-children-def best-children-def is-arg-max-def*)
    apply (*simp add*: *maximum-on-non-strict-def upper-bound-on-non-strict-def*)
    apply *auto*
    by (*smt children-type ex-in-conv subsetCE*)
qed

definition (**in** *BlockchainParams*) *best-child* :: *consensus-value* $\Rightarrow$ *state-property*
  where
    best-child b = ($\lambda\sigma$. $b \in$ *best-children* (*prev* b, σ))

function (**in** *BlockchainParams*) *GHOST* :: (*consensus-value set* $*$ *state*) $\Rightarrow$ *consensus-value set*
  where
    GHOST (*b-set*, σ) =

$(\bigcup$ *b* $\in \{$*b* $\in$ *b-set*. *children* $($*b*, $\sigma) \neq \emptyset\}$. *GHOST* $($*best-children* $($*b*, $\sigma)$, $\sigma))$
$\cup \{$*b* $\in$ *b-set*. *children* $($*b*, $\sigma) = \emptyset\}$
by *auto*

definition (**in** *BlockchainParams*) *GHOST-heads-or-children* :: *state* $\Rightarrow$ *consensus-value set*
where
GHOST-heads-or-children σ = *GHOST* $(\{$*genesis*$\}, \sigma) \cup (\bigcup$ *b* $\in$ *GHOST* $(\{$*genesis*$\}, \sigma)$. *children* $($*b*, $\sigma))$

lemma (**in** *Blockchain*) *GHOST-type* :
$\forall$ σ *b-set*. $\sigma \in \Sigma \wedge$ *b-set* $\subseteq C \longrightarrow$ *GHOST* $($*b-set*, $\sigma) \subseteq C$
proof −

have $\forall$ σ *b-set*. $\sigma \in \Sigma \wedge$ *b-set* $\subseteq C \longrightarrow (\exists$ *b-set'*. *b-set'* $\subseteq C \wedge$ *GHOST* $($*b-set*, $\sigma) = \{$*b* $\in$ *b-set'*. *children* $($*b*, $\sigma) = \emptyset\})$
sorry
then show *?thesis*
by *blast*
qed

lemma (**in** *Blockchain*) *GHOST-is-valid-estimator* :
is-valid-estimator GHOST-heads-or-children
unfolding *is-valid-estimator-def*
apply (*simp add*: *BlockchainParams.GHOST-heads-or-children-def*)
apply *auto*
using *GHOST-type genesis-type*(*1*) **apply** *blast*
using *GHOST-type children-type genesis-type*(*1*) **apply** *blast*
using *best-children-existence*
oops

locale *TFG* = *Blockchain* +
assumes *ghost-estimator* : ε = *GHOST-heads-or-children*

lemma (**in** *TFG*) *block-membership-is-majority-driven* :
$\forall$ *b* $\in C$. *majority-driven* (*block-membership b*)
apply (*simp add*: *majority-driven-def*)
oops

lemma (**in** *Blockchain*) *agreeing-validators-on-sistor-blocks-are-disagreeing* :
$\forall$ $\sigma \in \Sigma$. $\forall$ *b b1 b2*. $\{$*b*, *b1*, *b2*$\} \subseteq C \wedge \{$*b1*, *b2*$\} \subseteq$ *children* $($*b*, $\sigma)$
$\longrightarrow$ *agreeing-validators* (*block-membership b1*, $\sigma) \subseteq$ *disagreeing-validators* (*block-membership b2*, σ)
proof −
have $\forall$ $\sigma \in \Sigma$. $\forall$ *b b1 b2*. $\{$*b*, *b1*, *b2*$\} \subseteq C \wedge \{$*b1*, *b2*$\} \subseteq$ *children* $($*b*, $\sigma)$
$\longrightarrow$ $(\forall$ *v* $\in$ *agreeing-validators* (*block-membership b1*, σ). $\forall$ *c* $\in$*L-H-E* σ *v*.

```
block-membership b1 c)
    by (simp add: agreeing-validators-def agreeing-def)
  hence ∀ σ ∈ Σ. ∀ b b1 b2. {b, b1, b2} ⊆ C ∧ {b1, b2} ⊆ children (b, σ)
   ⟶ (∀ v ∈ agreeing-validators (block-membership b1, σ). ∃ c ∈L-H-E σ v. ¬
block-membership b2 c)
    using children-conflicting
    apply (simp add: block-membership-def block-conflicting-def)
    using irreflexivity-of-blockchain-membership by fast
  then show ?thesis
    using disagreeing-validators-include-not-agreeing-validators
    by (metis (no-types, lifting) ‹∀σ∈Σ. ∀ b b1 b2. {b, b1, b2} ⊆ C ∧ {b1, b2} ⊆
children (b, σ) ⟶ (∀ v∈agreeing-validators (block-membership b1, σ). ∀ c∈L-H-E
σ v. block-membership b1 c)› insert-subset subsetI)
qed

lemma (in Blockchain) agreeing-validators-on-sistor-blocks-are-not-more-than-disagreeing
:
  ∀ σ ∈ Σ. ∀ b b1 b2. {b, b1, b2} ⊆ C ∧ {b1, b2} ⊆ children (b, σ)
  ⟶ weight-measure (agreeing-validators (block-membership b1, σ)) ≤ weight-measure
(disagreeing-validators (block-membership b2, σ))
  using agreeing-validators-on-sistor-blocks-are-disagreeing
      agreeing-validators-on-sistor-blocks-are-disagreeing weight-measure-subset-gte
       agreeing-validators-type disagreeing-validators-type
  by auto

lemma (in Blockchain) no-child-and-best-child-at-all-earlier-height-imps-GHOST-heads
:
  ∀ σ ∈ Σ. ∀ b ∈ C. children (b, σ) = ∅ ∧
    (∀ b′ ∈ C. b′ ⇂ b ⟶ b′ ∈ best-children (prev b′, σ))
    ⟶ b ∈ GHOST ({genesis}, σ)
  apply auto
  oops

lemma (in Blockchain) best-child-at-all-earlier-height-imps-GHOST-heads-or-decendant
:
  ∀ σ ∈ Σ. ∀ b ∈ C.
    (∀ b′ ∈ C. b′ ⇂ b ⟶ b′ ∈ best-children (prev b′, σ))
    ⟶ (∀ b″ ∈ GHOST ({genesis}, σ). b ⇂ b″)
proof −
  have ⋀ n. ∀ σ ∈ Σ. ∀ b ∈ C. genesis = n-cestor (b, n) ∧
    (∀ b′ ∈ C. b′ ⇂ b ⟶ b′ ∈ best-children (prev b′, σ))
    ⟶ (∀ b″ ∈ GHOST ({genesis}, σ). b ⇂ b″)
  proof −
    fix n
    show ∀σ∈Σ. ∀ b∈C. genesis = n-cestor (b, n) ∧
                    (∀ b′ ∈ C. b′ ⇂ b ⟶ b′ ∈ best-children (prev b′, σ)) ⟶
                    (∀ b″∈GHOST ({genesis}, σ). b ⇂ b″)
      apply (induction n)
      using genesis-type GHOST-type
      apply (metis contra-subsetD empty-subsetI insert-subset n-cestor.simps(1))
    proof -
      fix n
      assume ∀σ∈Σ. ∀b∈C. genesis = n-cestor (b, n) ∧
                  (∀ b' ∈ C. b' ⌊ b ⟶ b' ∈ best-children (prev b', σ)) ⟶
                  (∀b''∈GHOST ({genesis}, σ). b ⌊ b'')
      show ∀σ∈Σ. ∀b∈C. genesis = n-cestor (b, Suc n) ∧
                  (∀ b' ∈ C. b' ⌊ b ⟶ b' ∈ best-children (prev b', σ)) ⟶
                  (∀b''∈GHOST ({genesis}, σ). b ⌊ b'')
       apply (rule, rule, rule, rule)
      proof -
       fix σ b b''
       assume σ ∈ Σ
       and b ∈ C
      and genesis = n-cestor (b, Suc n) ∧ (∀b' ∈ C. b' ⌊ b ⟶ b' ∈ best-children
(prev b', σ))
      and b'' ∈ GHOST ({genesis}, σ)
       then have genesis = n-cestor (prev b, n) ∧ (∀ b' ∈ C. b' ⌊ prev b ⟶ b'
∈ best-children (prev b', σ))
          by (metis BlockchainParams.blockchain-membership-def Blockchain-
Params.n-cestor.simps(2) diff-Suc-1 id-apply of-nat-eq-id of-nat-in-Nats)
       then have prev b ⌊ b''
         using ‹∀σ∈Σ. ∀ b ∈ C. genesis = n-cestor (b, n) ∧
                  (∀ b' ∈ C. b' ⌊ b ⟶ b' ∈ best-children (prev b', σ)) ⟶
                  (∀b''∈GHOST ({genesis}, σ). b ⌊ b'')›
         using ‹σ ∈ Σ› ‹b ∈ C› prev-type ‹b'' ∈ GHOST ({genesis}, σ)› by auto
       have b ∈ best-children (prev b, σ)
          using ‹genesis = n-cestor (b, Suc n) ∧ (∀ b' ∈ C. b' ⌊ b ⟶ b' ∈
best-children (prev b', σ))›
         using ‹b ∈ C› irreflexivity-of-blockchain-membership by blast
       then show b ⌊ b''
         using ‹prev b ⌊ b''› ‹b'' ∈ GHOST ({genesis}, σ)›
         sorry
      qed
    qed
  qed
  then show ?thesis
    using blockchain-membership-def genesis-type(2) by auto
qed

lemma (in TFG) ancestor-of-observed-block-is-observed :
    ∀ σ ∈ Σ. ∀ b ∈ est ‘σ. ∀ b' ∈ C. b' ⌊ b ⟶ b' ∈ est ‘σ
  sorry

lemma (in TFG) block-membership-is-max-driven :
  ∀ σ ∈ Σ. ∀ b ∈ est ‘σ. max-driven-for-future (block-membership b) σ
  apply (simp add: max-driven-for-future-def)
proof -
  have ∀ σ ∈ Σ. ∀ b b'. {b, b'} ⊆ C ∧ b' ⌊ b
```

$\longrightarrow$ *agreeing-validators* (*block-membership* b, σ) $\subseteq$ *agreeing-validators* (*block-membership* b', σ)
unfolding *agreeing-validators-def*
using *also-agreeing-on-ancestors* **by** *blast*
hence $\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b,\ b'\} \subseteq C \wedge b' \downharpoonleft b$
$\longrightarrow$ *weight-measure* (*agreeing-validators* (*block-membership* b', σ)) $\geq$ *weight-measure* (*agreeing-validators* (*block-membership* b, σ))
using *weight-measure-subset-gte agreeing-validators-finite agreeing-validators-type*
by *simp*
hence $\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b,\ b'\} \subseteq C \wedge b' \downharpoonleft b$
$\longrightarrow$ *weight-measure* V $-$ *weight-measure* (*disagreeing-validators* (*block-membership* b', σ)) $-$ *equivocation-fault-weight* σ
$\geq$ *weight-measure* V $-$ *weight-measure* (*disagreeing-validators* (*block-membership* b, σ)) $-$ *equivocation-fault-weight* σ
using *agreeing-validators-weight-combined* **by** *simp*
hence $\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b,\ b'\} \subseteq C \wedge b' \downharpoonleft b$
$\longrightarrow$ *weight-measure* (*disagreeing-validators* (*block-membership* b, σ))
$\geq$ *weight-measure* (*disagreeing-validators* (*block-membership* b', σ))
by *simp*
show $\forall\ \sigma \in \Sigma.\ \forall\ m \in \sigma.\ \forall\ \sigma' \in \Sigma.\ \sigma \subseteq \sigma' \longrightarrow$ *weight-measure* (*disagreeing-validators* (*block-membership* (*est m*), σ')) $<$ *weight-measure* (*agreeing-validators* (*block-membership* (*est m*), σ'))
$\longrightarrow$ ($\forall\ c \in \varepsilon\ \sigma'$. *block-membership* (*est m*) c)
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof $-$
fix $\sigma\ m\ \sigma'\ c$
assume $\sigma \in \Sigma$
and $m \in \sigma$
and $\sigma' \in \Sigma$
and $\sigma \subseteq \sigma'$
and *weight-measure* (*disagreeing-validators* (*block-membership* (*est m*), σ')) $<$ *weight-measure* (*agreeing-validators* (*block-membership* (*est m*), σ'))
and $c \in \varepsilon\ \sigma'$
hence *est m* $\in C$
using *M-type message-in-state-is-valid* **by** *blast*
hence $\forall\ b' \in C.\ b' \downharpoonleft$ *est m* $\longrightarrow$ *weight-measure* (*agreeing-validators* (*block-membership* b', σ')) $>$ *weight-measure* (*disagreeing-validators* (*block-membership* (*est m*), σ'))
using ‹$\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b,\ b'\} \subseteq C \wedge b' \downharpoonleft b$
$\longrightarrow$ *weight-measure* (*agreeing-validators* (*block-membership* b', σ)) $\geq$ *weight-measure* (*agreeing-validators* (*block-membership* b, σ))›
‹*weight-measure* (*disagreeing-validators* (*block-membership* (*est m*), σ')) $<$ *weight-measure* (*agreeing-validators* (*block-membership* (*est m*), σ'))›
‹$\sigma' \in \Sigma$› **by** *fastforce*
hence $\forall\ b' \in C.\ b' \downharpoonleft$ *est m* $\longrightarrow$ *weight-measure* (*agreeing-validators* (*block-membership* b', σ')) $>$ *weight-measure* (*disagreeing-validators* (*block-membership* b', σ'))
using ‹$\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b,\ b'\} \subseteq C \wedge b' \downharpoonleft b$
$\longrightarrow$ *weight-measure* (*disagreeing-validators* (*block-membership* b, σ)) $\geq$ *weight-measure* (*disagreeing-validators* (*block-membership* b', σ))›
‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$› ‹*est m* $\in C$› **by** *force*

```
    have ∀ b′ ∈ C. b′ ↓ est m ⟶ b′ ∈ best-children (prev b′, σ′)
      apply (simp add: best-children-def is-arg-max-def score-def)
      apply (auto)
      using ancestor-of-observed-block-is-observed
    apply (meson ‹σ ⊆ σ′› ‹σ′ ∈ Σ› ‹m ∈ σ› contra-subsetD image-eqI observed-block-is-children-of-prev-block)

      using M-type Params.message-in-state-is-valid ‹σ ∈ Σ›
      using agreeing-validators-on-sistor-blocks-are-not-more-than-disagreeing
            prev-type
        ‹∀ b′ ∈ C. b′ ↓ est m ⟶ weight-measure (agreeing-validators (block-membership
b′, σ′)) > weight-measure (disagreeing-validators (block-membership b′, σ′))›
    by (smt ‹σ′ ∈ Σ› agreeing-validators-weight-combined children-type contra-subsetD
empty-subsetI insert-absorb2 insert-subset)
    have c ∈ GHOST ({genesis}, σ′) ∪ (⋃ b ∈ GHOST ({genesis}, σ′). children
(b, σ′))
      using ghost-estimator ‹c ∈ ε σ′›
      unfolding GHOST-heads-or-children-def
      by blast
    have ∀ b″ ∈ GHOST ({genesis}, σ′). est m ↓ b″
       using best-child-at-all-earlier-height-imps-GHOST-heads-or-decendant ‹∀ b′
∈ C. b′ ↓ est m ⟶ b′ ∈ best-children (prev b′, σ′)›
            ‹σ ∈ Σ› ‹σ′ ∈ Σ› ‹est m ∈ C› by blast
   then show block-membership (est m) c
     unfolding block-membership-def
    using ‹c ∈ GHOST ({genesis}, σ′) ∪ (⋃ b ∈ GHOST ({genesis}, σ′). children
(b, σ′))›
           transitivity-of-blockchain-membership children-membership
     by blast
 qed
qed

end
```