Minimal CBC Casper Isabelle/HOL proofs

LayerX

August 2, 2019

Contents

theory *Strict-Order*

imports *Main*

begin

notation *Set.empty* (∅)

definition *strict-partial-order r* ≡ *trans r* ∧ *irrefl r*

definition *strict-well-order-on A r* ≡ *strict-linear-order-on A r* ∧ *wf r*

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

definition *upper-bound-on* :: ′*a set* ⇒ ′*a rel* ⇒ ′*a* ⇒ *bool*
  where
    upper-bound-on A r x = (∀ *y*. *y* ∈ *A* ⟶ (*y*, *x*) ∈ *r* ∨ *x* = *y*)

definition *maximum-on* :: ′*a set* ⇒ ′*a rel* ⇒ ′*a* ⇒ *bool*
  where

maximum-on A r x = ($x \in A \wedge$ *upper-bound-on A r x*)

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

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

lemma *maximal-and-maximum-coincide-for-strict-linear-order* :
strict-linear-order-on A r $\Longrightarrow$ *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 $\longrightarrow$ *finite A* $\longrightarrow A \neq \emptyset \longrightarrow$ ($\exists\ x.$ *maximal-on A r x*)
proof −
have $\bigwedge n$. *strict-partial-order r* $\Longrightarrow$ ($\forall\ A.$ *Suc n* = *card A* $\longrightarrow$ *finite A* $\longrightarrow A \neq \emptyset \longrightarrow$ ($\exists\ x.$ *maximal-on A r x*))
proof −
assume *strict-partial-order r*
then have ($\forall a.\ (a,\ a) \notin r$)
by (*simp add*: *strict-partial-order-def irrefl-def*)
fix *n*
show $\forall\ A.$ *Suc n* = *card A* $\longrightarrow$ *finite A* $\longrightarrow A \neq \emptyset \longrightarrow$ ($\exists\ x.$ *maximal-on A r x*)
apply (*induction n*)
unfolding *maximal-on-def*
using ‹($\forall a.\ (a,\ a) \notin r$)›
apply (*metis card-eq-SucD empty-iff insert-iff*)
proof −
fix *n*
assume $\forall A.$ *Suc n* = *card A* $\longrightarrow$ *finite A* $\longrightarrow A \neq \emptyset \longrightarrow (\exists x.\ x \in A \wedge (\forall y.\ (x,\ y) \in r \longrightarrow y \notin A))$
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* $\longrightarrow B \neq \emptyset$
$\longrightarrow (\exists\ A'\ b\ x.\ B = A' \cup \{b\} \wedge b \notin A' \wedge x \in A' \wedge (\forall y.\ (x,\ y) \in r \longrightarrow y \notin A'))$
using ‹$\forall A.$ *Suc n* = *card A* $\longrightarrow$ *finite A* $\longrightarrow A \neq \emptyset \longrightarrow (\exists x.\ x \in A \wedge (\forall y.\ (x,\ y) \in r \longrightarrow y \notin A))$›
by *metis*

then show $\forall B.\ Suc\ (Suc\ n) = card\ B \longrightarrow finite\ B \longrightarrow B \neq \emptyset \longrightarrow (\exists x.\ x \in B \wedge (\forall y.\ (x, y) \in r \longrightarrow y \notin B))$
by (*metis* (*no-types*, *lifting*) *Un-insert-right* ‹$\forall a.\ (a, a) \notin r$› ‹*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
$\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset$
$\longrightarrow is\text{-}singleton\ \{x.\ maximum\text{-}on\ A\ r\ x\}$
proof (*rule ccontr*)
assume $\neg$ (*strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow$ *is-singleton* $\{x.\ maximum\text{-}on\ A\ r\ x\}$)
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow \neg$ *is-singleton* $\{x.\ maximum\text{-}on\ A\ r\ x\}$
by *simp*
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow (\exists\ x1\ x2.\ x1 \neq x2 \wedge \{x1, x2\} \subseteq \{x.\ maximum\text{-}on\ A\ r\ x\})$
by (*meson empty-subsetI insert-subset is-singletonI'*)
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow (\exists\ x1\ x2.\ x1 \neq x2 \wedge \{x1, x2\} \subseteq \{x \in A.\ \forall\ y.\ y \in A \longrightarrow (y, x) \in r \vee x = y\})$
by (*simp add*: *maximum-on-def upper-bound-on-def*)
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow (\exists\ x1\ x2.\ x1 \neq x2 \wedge \{x1, x2\} \subseteq A \wedge (\forall\ y.\ y \in A \longrightarrow (y, x1) \in r \vee x1 = y) \wedge (\forall\ y.\ y \in A \longrightarrow (y, x2) \in r \vee x2 = y))$
by *auto*
then show *False*
using *strict-partial-order-def*

by (*metis* ‹$\neg$ (*strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow$ *is-singleton* $\{x.\ maximum\text{-}on\ A\ r\ x\}$)› *insert-subset irrefl-def transE*)
qed

lemma *strict-linear-order-on-finite-non-empty-set-has-one-maximum* :
strict-linear-order-on A r $\longrightarrow$ *finite A* $\longrightarrow A \neq \emptyset \longrightarrow$ *is-singleton* $\{x.\ maximum\text{-}on\ A\ r\ x\}$
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
    $\Sigma i\ (V,C,\varepsilon)\ 0 = \{\emptyset\}$
  $\mid \Sigma i\ (V,C,\varepsilon)\ n = \{\sigma \in Pow\ (Mi\ (V,C,\varepsilon)\ (n - 1)).\ finite\ \sigma \wedge (\forall\ m.\ m \in \sigma \longrightarrow justification\ m \subseteq \sigma)\}$
  $\mid Mi\ (V,C,\varepsilon)\ n = \{m.\ est\ m \in C \wedge sender\ m \in V \wedge justification\ m \in (\Sigma i\ (V,C,\varepsilon)\ n) \wedge est\ m \in \varepsilon\ (justification\ m)\}$

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: $Mi\ (V,C,\varepsilon)\ n \subseteq Mi\ (V,C,\varepsilon)\ (n{+}1) \Longrightarrow \Sigma i\ (V,C,\varepsilon)$

$(n+1) \subseteq \Sigma i\ (V,C,\varepsilon)\ (n+2)$
by *auto*

lemma *Σi-monotonic*: $\Sigma i\ (V,C,\varepsilon)\ n \subseteq \Sigma i\ (V,C,\varepsilon)\ (n+1)$
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*: $Mi\ (V,C,\varepsilon)\ n \subseteq Mi\ (V,C,\varepsilon)\ (n+1)$
apply (*induction n*)
defer
using *Σi-monotonic Σi-subset-to-Mi* **apply** *blast*
apply *auto*
done

lemma *Σi-monotonicity*: $\forall\ m \in \mathbb{N}.\ \forall\ n \in \mathbb{N}.\ m \leq n \longrightarrow \Sigma i\ (V,C,\varepsilon)\ m \subseteq \Sigma i\ (V,C,\varepsilon)\ n$
using *Σi-monotonic*
by (*metis Suc-eq-plus1 lift-Suc-mono-le*)

lemma *Mi-monotonicity*: $\forall\ m \in \mathbb{N}.\ \forall\ n \in \mathbb{N}.\ m \leq n \longrightarrow Mi\ (V,C,\varepsilon)\ m \subseteq Mi\ (V,C,\varepsilon)\ n$
using *Mi-monotonic*
by (*metis Suc-eq-plus1 lift-Suc-mono-le*)

lemma *message-is-in-Mi* :
$\forall\ m \in M.\ \exists\ n \in \mathbb{N}.\ m \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
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* :
$\forall\ \sigma \in \Sigma.\ (\exists\ n \in \mathbb{N}.\ \sigma \in Pow\ (Mi\ (V,\ C,\ \varepsilon)\ (n - 1)) \wedge (\forall\ m \in \sigma.\ justification\ m \subseteq \sigma))$
apply (*simp add*: *Σ-def*)

apply *auto*
proof –
fix y :: *nat* **and** σ :: *message set*
assume *a1*: $\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ y$
assume *a2*: $y \in \mathbb{N}$
have $\sigma \subseteq Mi\ (V,\ C,\ \varepsilon)\ y$
using *a1* **by** (*meson Params.Σi-monotonic Params.Σi-subset-Mi Pow-iff contra-subsetD*)
then have $\exists n.\ n \in \mathbb{N} \wedge \sigma \subseteq Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
using *a2* **by** (*metis* (*no-types*) *Nats-1 Nats-add diff-Suc-1 plus-1-eq-Suc*)
then show $\exists n{\in}\mathbb{N}.\ \sigma \subseteq \{m.\ est\ m \in C \wedge sender\ m \in V \wedge justification\ m \in \Sigma i\ (V,\ C,\ \varepsilon)\ (n - Suc\ 0) \wedge est\ m \in \varepsilon\ (justification\ m)\}$

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* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \land m \in \sigma \longrightarrow\ m \in M$
apply (*rule*, *rule*, *rule*)
proof −
fix σ m
assume $\sigma \in \Sigma \land m \in \sigma$
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* ‹$\sigma \in \Sigma \land m \in \sigma$› *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* $m \in C \land$ *sender* $m \in V \land$ *justification* $m \in \Sigma$
unfolding *M-def Σ-def*
by *auto*

end

locale *Protocol* = *Params* +
  assumes *V-type*: $V \neq \emptyset \wedge$ *finite V*
  and *W-type*: $\forall\ v \in V.\ W\ v > 0$
  and *t-type*: $0 \leq t\ t <$ *sum W V*
  and *C-type*: *card C* > 1
  and ε-*type*: *is-valid-estimator* ε

lemma (**in** *Protocol*) *estimates-are-non-empty*: $\bigwedge \sigma.\ \sigma \in \Sigma \Longrightarrow \varepsilon\ \sigma \neq \emptyset$
  using *is-valid-estimator-def* ε-*type* **by** *auto*

lemma (**in** *Protocol*) *estimates-are-subset-of-C*: $\bigwedge \sigma.\ \sigma \in \Sigma \Longrightarrow \varepsilon\ \sigma \subseteq C$
  using *is-valid-estimator-def* ε-*type* **by** *auto*

lemma (**in** *Params*) *empty-set-exists-in-*Σ*-0*: $\emptyset \in \Sigma i\ (V,\ C,\ \varepsilon)\ 0$
  by *simp*

lemma (**in** *Params*) *empty-set-exists-in-*Σ: $\emptyset \in \Sigma$
  apply (*simp add*: Σ-*def*)
  using *Nats-0* Σi.*simps*(*1*) **by** *blast*

lemma (**in** *Params*) Σi-*is-non-empty*: $\Sigma i\ (V,\ C,\ \varepsilon)\ n \neq \emptyset$
  apply (*induction n*)
  using *empty-set-exists-in-*Σ*-0* **by** *auto*

lemma (**in** *Params*) Σ*is-non-empty*: $\Sigma \neq \emptyset$
  using *empty-set-exists-in-*Σ **by** *blast*

lemma (**in** *Protocol*) *estimates-exists-for-empty-set* :
  $\varepsilon\ \emptyset \neq \emptyset$
  by (*simp add*: *empty-set-exists-in-*Σ *estimates-are-non-empty*)

lemma (**in** *Protocol*) *non-justifying-message-exists-in-M-0*:
  $\exists\ m.\ m \in Mi\ (V,\ C,\ \varepsilon)\ 0 \wedge$ *justification* $m = \emptyset$
  apply *auto*
proof −
  have $\varepsilon\ \emptyset \subseteq C$
    using *Params.empty-set-exists-in-*Σ ε-*type is-valid-estimator-def* **by** *auto*
  then show $\exists\, m.$ *est* $m \in C \wedge$ *sender* $m \in V \wedge$ *justification* $m = \emptyset \wedge$ *est* $m \in \varepsilon$ (*justification m*) $\wedge$ *justification* $m = \emptyset$
    by (*metis V-type all-not-in-conv est.simps estimates-exists-for-empty-set justification.simps sender.simps set-empty subsetCE*)
qed

lemma (**in** *Protocol*) *Mi-is-non-empty*: $Mi\ (V,\ C,\ \varepsilon)\ n \neq \emptyset$
  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* :
$\forall\ n \in \mathbb{N}.\ (\forall\ \sigma.\ \sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n \longrightarrow (\exists\ m.\ m \in Mi\ (V,\ C,\ \varepsilon)\ n \wedge \mathit{justification}\ m = \sigma))$
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* $\langle\sigma \in \Sigma\rangle$ **by** *auto*
have *finite* σ
using *state-is-finite* $\langle\sigma \in \Sigma\rangle$ **by** *auto*
moreover have $\exists\ m.\ \mathit{sender}\ m \in V \wedge \mathit{est}\ m \in \varepsilon\ \sigma \wedge \mathit{justification}\ m = \sigma$
using *est.simps sender.simps justification.simps V-type* $\langle\varepsilon\ \sigma \neq \emptyset\rangle$ $\langle\mathit{finite}\ \sigma\rangle$
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 $\langle n \in \mathbb{N}\rangle$ $\langle\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n\rangle$ **by** *blast*
ultimately show $\exists\ m.\ \mathit{est}\ m \in C \wedge \mathit{sender}\ m \in V \wedge \mathit{justification}\ m \in \Sigma i\ (V,\ C,\ \varepsilon)\ n \wedge \mathit{est}\ m \in \varepsilon\ (\mathit{justification}\ m) \wedge \mathit{justification}\ m = \sigma$
using *Nats-1 One-nat-def*
using $\langle\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n\rangle$ **by** *blast*
qed

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

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

lemma (**in** *Protocol*) *M-type-counterexample*:
$(\forall\ \sigma.\ \varepsilon\ \sigma = C) \Longrightarrow M = \{m.\ est\ m \in C \wedge sender\ m \in V \wedge justification\ m \in \Sigma\}$
  apply (*simp add*: *M-def*)
  apply *auto*
  using Σ*i-is-subset-of-*Σ **apply** *blast*
  by (*simp add*: Σ*-def*)

definition *observed* :: *message set* $\Rightarrow$ *validator set*
  where
    observed $\sigma = \{sender\ m \mid m.\ m \in \sigma\}$

lemma (**in** *Protocol*) *observed-type* :
$\forall\ \sigma \in Pow\ M.\ observed\ \sigma \in Pow\ V$
  using *Params.M-type Protocol-axioms observed-def* **by** *fastforce*

lemma (**in** *Protocol*) *observed-type-for-state* :
$\forall\ \sigma \in \Sigma.\ observed\ \sigma \subseteq V$
  using *Params.M-type Protocol-axioms observed-def state-is-subset-of-M* **by** *fastforce*

fun *is-future-state* :: (*state* $*$ *state*) $\Rightarrow$ *bool*
  where
    is-future-state $(\sigma 1,\ \sigma 2) = (\sigma 1 \subseteq \sigma 2)$

lemma (**in** *Params*) *state-difference-is-valid-message* :
$\forall\ \sigma\ \sigma'.\ \sigma \in \Sigma \wedge \sigma' \in \Sigma$
$\longrightarrow$ *is-future-state*$(\sigma,\ \sigma')$
$\longrightarrow \sigma' - \sigma \subseteq M$
  using *state-is-subset-of-M* **by** *blast*

definition *justified* :: *message* $\Rightarrow$ *message* $\Rightarrow$ *bool*
  where
    justified m1 m2 = (*m1* $\in$ *justification m2*)

definition *equivocation* :: (*message* ∗ *message*) ⇒ *bool*
where
equivocation =
(λ(*m1*, *m2*). *sender m1* = *sender m2* ∧ *m1* ≠ *m2* ∧ ¬ (*justified m1 m2*) ∧ ¬ (*justified m2 m1*))

definition *is-equivocating* :: *state* ⇒ *validator* ⇒ *bool*
where
is-equivocating σ *v* = (∃ *m1* ∈ σ. ∃ *m2* ∈ σ. *equivocation* (*m1*, *m2*) ∧ *sender m1* = *v*)

definition *equivocating-validators* :: *state* ⇒ *validator set*
where
equivocating-validators σ = {*v* ∈ *observed* σ. *is-equivocating* σ *v*}

lemma (**in** *Protocol*) *equivocating-validators-type* :
∀ σ ∈ Σ. *equivocating-validators* σ ⊆ *V*
using *observed-type-for-state equivocating-validators-def* **by** *blast*

lemma (**in** *Protocol*) *equivocating-validators-is-finite* :
∀ σ ∈ Σ. *finite* (*equivocating-validators* σ)
using *V-type equivocating-validators-type rev-finite-subset* **by** *blast*

definition (**in** *Params*) *equivocating-validators-paper* :: *state* ⇒ *validator set*
where
equivocating-validators-paper σ = {*v* ∈ *V*. *is-equivocating* σ *v*}

lemma (**in** *Protocol*) *equivocating-validators-is-equivalent-to-paper* :
∀ σ ∈ Σ. *equivocating-validators* σ = *equivocating-validators-paper* σ
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* :
∀ σ σ′ *v*. *is-future-state* (σ, σ′) ∧ *v* ∈ *V*
⟶ *v* ∈ *equivocating-validators* σ
⟶ *v* ∈ *equivocating-validators* σ′
apply (*simp add*: *equivocating-validators-def is-equivocating-def*)
using *observed-def* **by** *fastforce*

lemma (**in** *Protocol*) *equivocating-validators-preserved-over-honest-message* :
∀ σ *m*. σ ∈ Σ ∧ *m* ∈ *M*
⟶ *sender m* ∉ *equivocating-validators* (σ ∪ {*m*})
⟶ *equivocating-validators* σ = *equivocating-validators* (σ ∪ {*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* ⟹ $A \subseteq B$
⟹ *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* ⟹ $A \subset B$
⟹ *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* ⟹ $A \cap B = \emptyset$
⟹ *weight-measure A* + *weight-measure B* = *weight-measure* ($A \cup B$)
apply (*simp add*: *weight-measure-def*)
by (*simp add*: *sum.union-disjoint*)

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

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

lemma (**in** *Protocol*) *weight-measure-subset-gte-diff* :
$A \subseteq V$ ⟹ $A \subseteq B$ ⟹ *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* :
$B \subseteq V$ ⟹ $A \subseteq B$ ⟹ *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* $\Rightarrow$ *real*
where

equivocation-fault-weight σ = *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* $\Rightarrow$ *bool*

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

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

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

lemma (**in** *Protocol*) *past-state-of-*Σt*-is-*Σt :
$\forall\ \sigma\ \sigma'.\ \sigma' \in \Sigma t \wedge \sigma \in \Sigma \wedge$ *is-future-state* (σ, σ')
$\longrightarrow \sigma \in \Sigma t$
using *equivocation-fault-weight-is-monotonic*
apply (*simp add*: Σt*-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* $m \in C$
assume *sender* $m \in V$
assume *justification* $m \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$
assume *est* $m \in \varepsilon$ (*justification* m)
assume $m \in$ *justification* m
have $m \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
by (*smt Mi.simps One-nat-def Params.Σi-subset-Mi Pow-iff Suc-pred* ‹*est* $m \in C$› ‹*est* $m \in \varepsilon$ (*justification* m)› ‹*justification* $m \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$› ‹$m \in$ *justification* m› ‹*sender* $m \in V$› *add.right-neutral add-Suc-right diff-is-0-eq′ diff-le-self diff-zero mem-Collect-eq not-gr0 subsetCE*)
then have *justification* $m \in \Sigma i\ (V,\ C,\ \varepsilon)\ (n - 1)$
using *Mi.simps* **by** *blast*
then have *justification* $m \in \Sigma i\ (V,\ C,\ \varepsilon)\ 0$
apply (*induction n*)
apply *simp*
by (*smt Mi.simps One-nat-def Params.Σi-subset-Mi Pow-iff Suc-pred* ‹$m \in$ *justification* m› *add.right-neutral add-Suc-right diff-Suc-1 mem-Collect-eq not-gr0 subsetCE subsetCE*)
then have *justification* $m \in \{\emptyset\}$
by *simp*
then show *False*
using ‹$m \in$ *justification* m› **by** *blast*
qed

lemma (**in** *Protocol*) *message-cannot-justify-itself* :
($\forall\ m \in M.\ \neg$ *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* :
$\forall\ m\ m'\ \sigma.\ m \in M \wedge \sigma \in \Sigma \wedge$ *justified* $m'\ m \longrightarrow$ *justification* $m' \subseteq$ *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* :
  $\forall\ m\ m'\ \sigma.\ m \in M \wedge \sigma \in \Sigma \wedge$ *justified* $m'\ m \longrightarrow$ *justification* $m' \subset$ *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* :
  $\forall\ m\ m'.\ m \in M \wedge m' \in M \longrightarrow$ *justified* $m'\ m \longrightarrow m \neq m'$
  using *message-cannot-justify-itself* **by** *auto*

lemma (**in** *Protocol*) *only-valid-message-is-justified* :
  $\forall\ m \in M.\ \forall\ m'.$ *justified* $m'\ m \longrightarrow m' \in M$
  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* :
  $\forall\ n\ m\ m'.\ n \in \mathbb{N}$
  $\longrightarrow$ *justified* $m'\ m$
  $\longrightarrow m \in Mi\ (V,\ C,\ \varepsilon)\ n$
  $\longrightarrow\ m' \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
proof −
  have $\forall\ n\ m\ m'.$ *justified* $m'\ m$
  $\longrightarrow m \in Mi\ (V,\ C,\ \varepsilon)\ n$
  $\longrightarrow m \in M \wedge m' \in M$
  $\longrightarrow m' \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
    apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
  proof −
    fix $n\ m\ m'$
    assume *justified* $m'\ m$
    assume $m \in Mi\ (V,\ C,\ \varepsilon)\ n$
    assume $m \in M \wedge m' \in M$
    then have *justification* $m \in \Sigma i\ (V,C,\varepsilon)\ n$
      using *Mi.simps* ‹$m \in Mi\ (V,\ C,\ \varepsilon)\ n$› **by** *blast*
    then have *justification* $m \in$ *Pow* $(Mi\ (V,C,\varepsilon)\ (n - 1))$
      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 $m' \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
      using ‹*justification* $m \in$ *Pow* $(Mi\ (V,\ C,\ \varepsilon)\ (n - 1))$› ‹*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* :
$\forall$ m m'. $m \in M$
$\longrightarrow$ *justified* m' m
$\longrightarrow$ *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 $\neg$ *wfp-on justified M*
then have $\exists f. \forall i.\ f\ i \in M \wedge$ *justified* (f (*Suc* i)) (f i)
by (*simp add*: *wfp-on-def*)
then obtain f **where** $\forall i.\ f\ i \in M \wedge$ *justified* (f (*Suc* i)) (f i) **by** *auto*
have $\forall$ i. *card* (*justification* (f i)) $\leq$ *card* (*justification* (f *0*)) $-$ i
apply (*rule*)
proof $-$
fix i
have *card* (*justification* (f (*Suc* i))) $<$ *card* (*justification* (f i))
using ‹$\forall i.\ f\ i \in M \wedge$ *justified* (f (*Suc* i)) (f i)› **by** (*simp add*: *monotonicity-of-card-of-justification*)
show *card* (*justification* (f i)) $\leq$ *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* ‹$\forall i.\ f\ i \in M \wedge$ *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 $\exists$ i. i = *card* (*justification* (f *0*)) + *Suc 0* $\wedge$ *card* (*justification* (f i)) $\leq$ *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* ‹$\forall i.\ f\ i \in M \wedge$ *justified* (f (*Suc* i)) (f i)› *add.right-neutral add-Suc-right*)
qed

lemma (**in** *Protocol*) *subset-of-M-have-minimal-of-justification* :
$\forall$ $S \subseteq M$. $S \neq \emptyset \longrightarrow$ ($\exists$ *m-min* $\in S$. $\forall$ m. *justified* m *m-min* $\longrightarrow m \notin S$)
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* :
$\forall$ $\sigma \in \Sigma$. $\forall$ $m \in \sigma$. *justification* $m \subset \sigma$
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 = $(\lambda(m, \sigma).\ \{m' \in \sigma.\ \mathit{justified}\ m\ m'\})$

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

lemma (**in** *Protocol*) *later-type-for-state* :
  $\forall\ \sigma\ m.\ \sigma \in \Sigma \wedge m \in M \longrightarrow \mathit{later}\ (m, \sigma) \subseteq M$
  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 = $(\lambda(v, \sigma).\ \{m \in \sigma.\ \mathit{sender}\ m = v\})$

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

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

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

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

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

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

lemma (**in** *Protocol*) *from-group-type-for-state* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge \textit{v-set} \subseteq V \longrightarrow \textit{from-group}\ (\textit{v-set},\ \sigma) \subseteq M$
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
$\textit{later-from} = (\lambda(m,\ v,\ \sigma).\ \{m' \in \sigma.\ \textit{sender}\ m' = v \wedge \textit{justified}\ m\ m'\})$

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

lemma (**in** *Protocol*) *later-from-type-for-state* :
$\forall\ \sigma\ v\ m.\ \sigma \in \Sigma \wedge v \in V \wedge m \in M \longrightarrow \textit{later-from}\ (m,\ v,\ \sigma) \subseteq M$
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
$\textit{L-M}\ \sigma\ v = \{m \in \textit{from-sender}\ (v,\ \sigma).\ \textit{later-from}\ (m,\ v,\ \sigma) = \emptyset\}$

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

lemma (**in** *Protocol*) *L-M-type-for-state* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow \textit{L-M}\ \sigma\ v \subseteq M$
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* $\sigma\ v = \{m.$ *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* $\sigma\ v$))

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* $\sigma\ v$)
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* :
  $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *L-E* σ $v \subseteq C$
  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* :
  $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \wedge v \notin$ *observed* $\sigma \longrightarrow$ *L-E* σ $v = \emptyset$
  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* $v \in$ *equivocating-validators* σ *then* $\emptyset$ *else L-M* σ v)

lemma (**in** *Protocol*) *L-H-M-type* :
  $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *L-H-M* σ $v \subseteq M$
  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* :
  $\forall\ \sigma \in \Sigma.\ \forall\ v \in$ *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*:
  $\forall\ \sigma \in \Sigma.\ \forall\ v \in$ *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*:
  $\forall\ \sigma \in \Sigma.\ \forall\ v \in$ *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* $\Rightarrow$ *validator* $\Rightarrow$ *consensus-value set*
where

L-H-E σ v = *est* '*L-H-M* σ v

lemma (**in** *Protocol*) *L-H-E-type* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *L-H-E* σ $v \in$ *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-H-E* σ $v = \emptyset$
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-singleton $A \Longrightarrow$ *is-singleton* (f '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-non-equivocating-validators* σ.
is-singleton (*L-H-E* σ 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* $\Rightarrow$ *validator* $\Rightarrow$ *state set*
where
L-H-J σ v = *justification* '*L-H-M* σ v

lemma (**in** *Protocol*) *L-H-J-type* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *L-H-J* σ $v \subseteq \Sigma$
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* :
$\forall\ \sigma \in \Sigma.\ v \in$ *observed-non-equivocating-validators* σ
$\longrightarrow$ *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* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \land v \in V \longrightarrow (\forall\ \sigma' \in$ *L-H-J* $\sigma\ v.\ \sigma' \subset \sigma)$
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 $= \{(\sigma 1,\ \sigma 2).\ \{\sigma 1,\ \sigma 2\} \subseteq \Sigma \land$ *is-future-state*$(\sigma 1,\ \sigma 2)\}$

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* $(\sigma,\ m)$

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* $(\sigma,\ m)$› *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* ‹$\exists n'. n = Suc\ n'$› *diff-Suc-1 subset-iff*)
ultimately have $\sigma \cup \{m\} \in Pow\ (Mi\ (V,C,\varepsilon)\ n)$
by *blast*

show $\sigma \cup \{m\} \in \Sigma i\ (V,\ C,\ \varepsilon)\ (n + 1)$
using ‹$\bigwedge m'.$ *finite* $\sigma \wedge m' \in \sigma \Longrightarrow$ *justification* $m' \subseteq \sigma \cup \{m\}$› ‹$\sigma \cup \{m\} \in Pow\ (Mi\ (V,\ C,\ \varepsilon)\ n)$› ‹*justification* $m \subseteq \sigma \cup \{m\}$›
‹$\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$› *si* **by** *auto*
qed

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

$\forall \sigma{\in}\Sigma i\ (V,C,\varepsilon)\ n.\ \forall m{\in}Mi\ (V,C,\varepsilon)\ n.$ *immediately-next-message* $(\sigma,m) \longrightarrow \sigma \cup \{m\} \in \Sigma i\ (V,C,\varepsilon)\ (n{+}1)$
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* :
$\forall\ \sigma\ \sigma'.\ \sigma' \in \Sigma i\ (V,C,\varepsilon)\ n \longrightarrow \sigma \subseteq \sigma' \longrightarrow \sigma \in \Sigma \longrightarrow \sigma \in \Sigma i\ (V,C,\varepsilon)\ n$
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof (*cases n*)
case *0*
show $\bigwedge\sigma\ \sigma'.\ \sigma' \in \Sigma i\ (V,\ C,\ \varepsilon)\ n \Longrightarrow \sigma \subseteq \sigma' \Longrightarrow \sigma \in \Sigma \Longrightarrow n = 0 \Longrightarrow \sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$
by *auto*
next
case (*Suc nat*)
show $\bigwedge\sigma\ \sigma'\ nat.\ \sigma' \in \Sigma i\ (V,\ C,\ \varepsilon)\ n \Longrightarrow \sigma \subseteq \sigma' \Longrightarrow \sigma \in \Sigma \Longrightarrow n = Suc\ nat \Longrightarrow \sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$
proof −
fix $\sigma\ \sigma'$
assume $\sigma' \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$
and $\sigma \subseteq \sigma'$
and $\sigma \in \Sigma$
have $n > 0$
by (*simp add*: *Suc*)
have *finite* $\sigma \wedge (\forall\ m.\ m \in \sigma \longrightarrow$ *justification* $m \subseteq \sigma)$
using ‹$\sigma \in \Sigma$› *state-is-finite state-is-in-pow-Mi* **by** *blast*
moreover have $\sigma \in Pow\ (Mi\ (V,\ C,\ \varepsilon)\ (n - 1))$
using ‹$\sigma \subseteq \sigma'$›
by (*smt Pow-iff Suc-eq-plus1* Σi*-monotonic* Σi*-subset-Mi* ‹$\sigma' \in \Sigma i\ (V,\ C,\ \varepsilon)\ n$› *add-diff-cancel-left′ add-eq-if diff-is-0-eq diff-le-self plus-1-eq-Suc subset-iff*)
ultimately have $\sigma \in \{\sigma \in Pow\ (Mi\ (V,C,\varepsilon)\ (n - 1)).$ *finite* $\sigma \wedge (\forall\ m.\ m \in \sigma \longrightarrow$ *justification* $m \subseteq \sigma)\}$
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 $\langle\neg\ (\exists\, m{\in}\sigma - \sigma'.$ *justification* $m \subseteq \sigma')\rangle$ *state-is-in-pow-Mi* $\langle\sigma' \subset \sigma\rangle$
by (*metis Diff-iff* $\langle\sigma \in \Sigma\rangle$ *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* $\langle\sigma' \subset \sigma\rangle$
by (*meson DiffD1* $\langle\sigma \in \Sigma\rangle$)
have $\sigma - \sigma' \subseteq M$
using $\langle\sigma \in \Sigma\rangle$ $\langle\sigma' \subset \sigma\rangle$ *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* $\langle\sigma' \subset \sigma\rangle$
by *blast*
then show *False*
using $\langle\forall\ m \in \sigma - \sigma'.\ \exists\ m'.$ *justified* $m'\ m \wedge m' \in \sigma - \sigma'\rangle$ **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* σ
assume $\sigma \subset \sigma'$

have $\exists\ m \in \sigma' - \sigma.$ *immediately-next-message* $(\sigma,\ m)$
using *strict-subset-of-state-have-immediately-next-messages*
$\langle\sigma \in \Sigma\rangle$ $\langle\sigma \subset \sigma'\rangle$ $\langle\sigma' \in$ *futures* $\sigma\rangle$
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* $\langle\sigma \in \Sigma\rangle$ $\langle\sigma' \in$ *futures* $\sigma\rangle$
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 $\langle\sigma \subset \sigma'\rangle$ **by** *auto*
then show $\exists m \in \sigma' - \sigma.\ \sigma \cup \{m\} \in \Sigma t$
using *equivocation-fault-weight-is-monotonic* $\langle\sigma' \in$ *futures* $\sigma\rangle$
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* $\sigma.\ \sigma \subset \sigma'$
$\longrightarrow (\exists\ m \in \sigma' - \sigma.$ *immediately-next-message* $(\sigma,\ m) \wedge \sigma \cup \{m\} \in \Sigma t \wedge \sigma' \in$ *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* $(\sigma,\ \sigma') \wedge \sigma \neq \sigma' \longrightarrow (\exists\ m \in \sigma' - \sigma.$ *immediately-next-message* $(\sigma,\ m))$
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* $\langle\sigma 2 \in \Sigma\rangle$)
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 ‹$\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in$

Σ)› ‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$› ‹$\neg\ \sigma \subseteq \sigma' \wedge$ *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* :
$\forall\ \sigma$-*set* $\subseteq \Sigma$. *finite* σ-*set* $\longrightarrow \bigcup\ \sigma$-*set* $\in \Sigma$
apply *auto*
proof −
have $\forall\ n.\ \forall\ \sigma$-*set* $\subseteq \Sigma$. n = *card* σ-*set* $\longrightarrow$ *finite* σ-*set* $\longrightarrow \bigcup\ \sigma$-*set* $\in \Sigma$
apply (*rule*)
proof −
fix n
show $\forall\ \sigma$-*set* $\subseteq \Sigma$. n = *card* σ-*set* $\longrightarrow$ *finite* σ-*set* $\longrightarrow \bigcup\sigma$-*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 $\forall\sigma$-*set*$\subseteq\Sigma$. n = *card* σ-*set* $\longrightarrow$ *finite* σ-*set* $\longrightarrow \bigcup\sigma$-*set* $\in \Sigma$ **and** σ-*set* $\subseteq \Sigma$ **and** *Suc* n = *card* σ-*set* **and** *finite* σ-*set*
then have $\forall\ \sigma \in \sigma$-*set*. σ-*set* $- \{\sigma\} \subseteq \Sigma \wedge \bigcup$ (σ-*set* $- \{\sigma\}$) $\in \Sigma$
using ‹σ-*set* $\subseteq \Sigma$› ‹*Suc* n = *card* σ-*set*› ‹$\forall\sigma$-*set*$\subseteq\Sigma$. n = *card* σ-*set* $\longrightarrow$ *finite* σ-*set* $\longrightarrow \bigcup\sigma$-*set* $\in \Sigma$›
by (*metis* (*mono-tags*, *lifting*) *Suc-inject card.remove finite-Diff insert-Diff insert-subset*)
then have $\forall\ \sigma \in \sigma$-*set*. σ-*set* $- \{\sigma\} \subseteq \Sigma \wedge \bigcup$ (σ-*set* $- \{\sigma\}$) $\in \Sigma \wedge \bigcup$ (σ-*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 $\bigwedge \sigma\text{-}set.\ \sigma\text{-}set \subseteq \Sigma \Longrightarrow \mathit{finite}\ \sigma\text{-}set \Longrightarrow \bigcup \sigma\text{-}set \in \Sigma$
by *blast*
qed

lemma (**in** *Protocol*) *non-empty-state-is-reached-by-receiving-single-message* :
$\forall\ \sigma \in \Sigma.\ \sigma \neq \emptyset \longrightarrow (\exists\ \sigma'\ m.\ \sigma' \in \Sigma \wedge m \in \sigma \wedge m \notin \sigma' \wedge \sigma = \sigma' \cup \{m\})$
sorry

lemma (**in** *Protocol*) *non-empty-state-is-reached-by-receiving-immediately-next-message* :
$\forall\ \sigma \in \Sigma.\ \sigma \neq \emptyset \longrightarrow (\exists\ \sigma'\ m.\ \sigma' \in \Sigma \wedge m \in \sigma \wedge \mathit{immediately\text{-}next\text{-}message}(\sigma', m) \wedge \sigma = \sigma' \cup \{m\})$
using *state-differences-have-immediately-next-messages*
state-transition-only-made-by-immediately-next-message
non-empty-state-is-reached-by-receiving-single-message
by (*metis message-in-state-is-valid*)

lemma (**in** *Protocol*) *intermediate-state-before-receiving-single-message* :
$\forall\ \sigma\ \sigma'.\ \{\sigma, \sigma'\} \subseteq \Sigma \wedge \sigma \subset \sigma' \wedge \sigma' \neq \emptyset$
$\longrightarrow (\exists\ \sigma''\ m.\ \sigma'' \in \Sigma \wedge m \in \sigma' \wedge \mathit{immediately\text{-}next\text{-}message}(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\} \wedge \sigma \subseteq \sigma'')$
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ \sigma'$
assume $\{\sigma, \sigma'\} \subseteq \Sigma \wedge \sigma \subset \sigma' \wedge \sigma' \neq \emptyset$
then have $\exists\ \sigma''\ m.\ \sigma'' \in \Sigma \wedge m \in \sigma' \wedge \mathit{immediately\text{-}next\text{-}message}(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\}$
using *non-empty-state-is-reached-by-receiving-immediately-next-message*
by *simp*
then obtain $\sigma''\ m$ **where** $\sigma'' \in \Sigma \wedge m \in \sigma' \wedge \mathit{immediately\text{-}next\text{-}message}(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\}$
by *auto*
then have $\sigma \subset \sigma' \wedge \sigma' \neq \emptyset \wedge \sigma' = \sigma'' \cup \{m\} \wedge m \in \sigma' \wedge m \notin \sigma''$
apply (*simp add*: *immediately-next-message-def*)
using ‹$\{\sigma, \sigma'\} \subseteq \Sigma \wedge \sigma \subset \sigma' \wedge \sigma' \neq \emptyset$› **by** *auto*
then have $\sigma \subseteq \sigma''$
sorry
then show $\exists \sigma''\ m.\ \sigma'' \in \Sigma \wedge m \in \sigma' \wedge \mathit{immediately\text{-}next\text{-}message}\ (\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\} \wedge \sigma \subseteq \sigma''$
using ‹$\sigma'' \in \Sigma \wedge m \in \sigma' \wedge \mathit{immediately\text{-}next\text{-}message}\ (\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\}$› **by** *blast*
qed

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* Σt*-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* (σ, *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 $\forall$ σ σ'. (σ, σ') $\in$ *minimal-transitions* $\wedge$ $\neg$ *is-singleton* ($\sigma' - \sigma$) $\longrightarrow$
        ($\exists$ σ''. $\sigma'' \in \Sigma$ $\wedge$ *is-future-state* (σ, σ'') $\wedge$ *is-future-state* (σ'', σ') $\wedge$ $\sigma \neq \sigma''$ $\wedge$ $\sigma'' \neq \sigma'$)
    apply (*rule*, *rule*, *rule*)
  proof −
    fix σ σ'
    assume (σ, σ') $\in$ *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* (σ, *m1*)
      using ‹$\forall$ σ σ'. (σ, σ') $\in$ *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* (σ, *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* (σ, *m1*)
      by *auto*
    have $\sigma \in \Sigma \wedge \sigma' \in \Sigma$
      using ‹(σ, σ') $\in$ *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* (σ, *m1*)›
        state-transition-by-immediately-next-message
      by *simp*
    have *is-future-state* (σ, $\sigma \cup$ {*m1*}) $\wedge$ *is-future-state* ($\sigma \cup$ {*m1*}, σ')
      using ‹(σ, σ') $\in$ *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* (σ, *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* (σ, *m1*)› **by** *auto*
    then show $\exists \sigma''$. $\sigma'' \in \Sigma$ $\wedge$ *is-future-state* (σ, σ'') $\wedge$ *is-future-state* (σ'', σ') $\wedge$ $\sigma \neq \sigma''$ $\wedge$ $\sigma'' \neq \sigma'$
      using ‹$\sigma \cup$ {*m1*} $\in \Sigma$› ‹*is-future-state* (σ, $\sigma \cup$ {*m1*}) $\wedge$ *is-future-state* ($\sigma \cup$ {*m1*}, σ')›
      by *auto*
  qed
  then show *False*
    using ‹$\forall \sigma$ σ'. (σ, σ') $\in$ *minimal-transitions* $\longrightarrow$ ($\nexists \sigma''$. $\sigma'' \in \Sigma$ $\wedge$ *is-future-state* (σ, σ'') $\wedge$ *is-future-state* (σ'', σ') $\wedge$ $\sigma \neq \sigma''$ $\wedge$ $\sigma'' \neq \sigma'$)› ‹$\neg$ ($\forall \sigma$ σ'. (σ, σ') $\in$ *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

lemma (**in** *Protocol*) *road-to-future-state* :
$\forall\ \sigma\ \sigma'.\ \sigma \in \Sigma \wedge \sigma' \in \Sigma \wedge$ *is-future-state*(σ, σ')
$\longrightarrow n =$ *card* $(\sigma' - \sigma)$
$\longrightarrow (\exists\ f.\ f\ 0 = \sigma \wedge f\ n = \sigma' \wedge (\forall\ i.\ 0 \leq i \wedge i \leq n - 1 \longrightarrow f\ i \in \Sigma \wedge (\exists\ m \in M.\ f\ i \cup \{m\} = f\ (Suc\ i))))$
apply (*rule*, *rule*, *rule*, *rule*)
oops

end

4 Safety Proof

theory *ConsensusSafety*

imports *Main CBCCasper MessageJustification StateTransition Libraries/LaTeXsugar*

begin

lemma (**in** *Protocol*) *monotonic-futures* :
$\forall\ \sigma'\ \sigma.\ \sigma' \in \Sigma t \wedge \sigma \in \Sigma t$
$\longrightarrow \sigma' \in \textit{futures}\ \sigma \longleftrightarrow \textit{futures}\ \sigma' \subseteq \textit{futures}\ \sigma$
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* $(\sigma 1 \cup \sigma 2)$
$\longrightarrow \textit{futures}\ \sigma 1 \cap \textit{futures}\ \sigma 2 \neq \emptyset$
apply (*simp add*: *futures-def* Σ*t-def*) **using** *union-of-two-states-is-state*
by *blast*

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

lemma (**in** *Protocol*) *n-party-common-futures-exists* :
$\forall\ \sigma\text{-}set.\ \sigma\text{-}set \subseteq \Sigma t$
$\longrightarrow \textit{finite}\ \sigma\text{-}set$
$\longrightarrow$ *is-faults-lt-threshold* $(\bigcup\ \sigma\text{-}set)$
$\longrightarrow (\exists\ \sigma \in \Sigma t.\ \sigma \in \bigcap\ \{\textit{futures}\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\})$
apply (*simp add*: *futures-def* Σ*t-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 $= (\lambda(p, \sigma).\ (\forall\ \sigma' \in \textit{futures}\ \sigma\ .\ p\ \sigma'))$

lemma (**in** *Protocol*) *forward-consistency* :
$\forall\ \sigma'\ \sigma.\ \sigma' \in \Sigma t \wedge \sigma \in \Sigma t$
$\longrightarrow \sigma' \in \textit{futures}\ \sigma$
$\longrightarrow$ *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 $p = (\lambda\sigma.\ (\neg\ p\ \sigma))$

lemma (**in** *Protocol*) *backword-consistency* :
$\forall\ \sigma'\ \sigma.\ \sigma' \in \Sigma t \land \sigma \in \Sigma t$
$\longrightarrow \sigma' \in$ *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* :
$\forall\ \sigma 1\ \sigma 2.\ \sigma 1 \in \Sigma t \land \sigma 2 \in \Sigma t$
$\longrightarrow$ *is-faults-lt-threshold* $(\sigma 1 \cup \sigma 2)$
$\longrightarrow \neg$(*state-property-is-decided* $(p, \sigma 1) \land$ *state-property-is-decided* (*state-property-not* p, $\sigma 2$))
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 $= (\forall\ \sigma \in \Sigma.\ \neg\ (\forall\ p \in$ *p-set*. $p\ \sigma))$

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

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

theorem (**in** *Protocol*) *n-party-safety-for-state-properties* :
$\forall\ \sigma$-*set*. σ-*set* $\subseteq \Sigma t$
$\longrightarrow$ *finite* σ-*set*
$\longrightarrow$ *is-faults-lt-threshold* $(\bigcup\ \sigma$-*set*$)$
$\longrightarrow$ *state-properties-are-consistent* $(\bigcup\ \{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$-*set*$\})$
apply *rule+*
proof−
fix σ-*set*
assume σ-*set*: σ-*set* $\subseteq \Sigma t$

and *finite* σ*-set*
and *is-faults-lt-threshold* ($\bigcup$ σ*-set*)
hence $\exists\sigma\in\Sigma t.\ \sigma \in \bigcap$ {*futures* σ | σ. $\sigma \in \sigma$*-set*}
using *n-party-common-futures-exists* **by** *simp*
hence $\exists\sigma\in\Sigma t.\ \forall s\in\sigma$*-set*. $\sigma \in$ *futures s*
by *blast*
hence $\exists\sigma\in\Sigma t.$ ($\forall s\in\sigma$*-set*. $\sigma \in$ *futures s*) $\wedge$ ($\forall s\in\sigma$*-set*. $\sigma \in$ *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 $\exists\sigma\in\Sigma t.\ \forall s\in\sigma$*-set*. ($\forall p$. *state-property-is-decided* (p,s) $\longrightarrow$ *state-property-is-decided* (p,σ))
by *blast*
hence $\exists\sigma\in\Sigma t.\ \forall s\in\sigma$*-set*. ($\forall p \in$ *state-property-decisions s*. *state-property-is-decided* (p,σ))
by (*simp add*: *state-property-decisions-def*)
hence $\exists\sigma\in\Sigma t.\ \forall p\in\bigcup$ {*state-property-decisions* σ | σ. $\sigma \in \sigma$*-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,σ))
using ‹$\exists\sigma\in\Sigma t.\ \forall s\in\sigma$*-set*. $\forall p\in$*state-property-decisions s*. *state-property-is-decided* (p, σ)› **by** *blast*
have $\forall p\in\bigcup$ {*state-property-decisions* σ | σ. $\sigma \in \sigma$*-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, σ)› **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* σ | σ. $\sigma \in \sigma$*-set*}. $\forall\sigma'\in$*futures* σ. $p\ \sigma'$
by (*simp add*: *state-property-decisions-def futures-def state-property-is-decided-def*)
show *state-properties-are-consistent* ($\bigcup$ {*state-property-decisions* σ |σ. $\sigma \in \sigma$*-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* σ |σ. $\sigma \in \sigma$*-set*}. $\forall\sigma'\in$*futures* σ. $p\ \sigma'$› *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* $q \mid q.\ q$
$\in$ *q-set*$\}$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ p \in \{\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q\ c \mid q.\ q \in$ *q-set*$\}.\ p\ \sigma)$
by (*simp add*: *naturally-corresponding-state-property-def state-properties-are-consistent-def*)
moreover have
$(\exists\ \sigma \in \Sigma.\ \forall\ p \in \{\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q\ c \mid q.\ q \in$ *q-set*$\}.\ p\ \sigma)$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ q' \in$ *q-set*. $(\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q'\ c)\ \sigma)$
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
moreover have
$(\exists\ \sigma \in \Sigma.\ \forall\ q \in$ *q-set*. $(\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q\ c)\ \sigma)$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ q' \in$ *q-set*. $\forall\ c \in \varepsilon\ \sigma.\ q'\ c)$
by *blast*
moreover have
$(\exists\ \sigma \in \Sigma.\ \forall\ q \in$ *q-set*. $\forall\ c \in \varepsilon\ \sigma.\ q\ c)$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ c \in \varepsilon\ \sigma.\ \forall\ q' \in$ *q-set*. $q'\ c)$
by *blast*
moreover have
$(\exists\ \sigma \in \Sigma.\ \forall\ c \in \varepsilon\ \sigma.\ \forall\ q \in$ *q-set*. $q\ c)$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \exists\ c \in \varepsilon\ \sigma.\ \forall\ q' \in$ *q-set*. $q'\ c)$
by (*meson all-not-in-conv estimates-are-non-empty*)
moreover have
$(\exists\ \sigma \in \Sigma.\ \exists\ c \in \varepsilon\ \sigma.\ \forall\ q \in$ *q-set*. $q\ c)$
$\longrightarrow (\exists\ c \in C.\ \forall\ q' \in$ *q-set*. $q'\ c)$
using *is-valid-estimator-def* ε*-type* **by** *fastforce*
ultimately show
state-properties-are-consistent $\{$*naturally-corresponding-state-property* $q \mid q.\ q \in$
q-set$\}$
$\Longrightarrow$ *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*) $\Rightarrow$ *bool*
where
consensus-value-property-is-decided
$= (\lambda(q, \sigma).$ *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* ⊆ Σ*t*
⟶ *finite* σ*-set*
⟶ *is-faults-lt-threshold* (⋃ σ*-set*)
⟶ *consensus-value-properties-are-consistent* (⋃ {*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* (⋃ σ*-set*)
hence *state-properties-are-consistent* (⋃ {*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* ∈ ⋃ {*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* ∈ ⋃ {*state-property-decisions* σ | σ. σ ∈ σ*-set*}}
by (*smt Collect-cong*)
hence *consensus-value-properties-are-consistent* {*q*. *naturally-corresponding-state-property q* ∈ ⋃ {*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*} ⟶ *consensus-value-properties-are-consistent q-set*› ‹*state-properties-are-consistent* {*naturally-corresponding-state-property q* |*q*. *naturally-corresponding-state-property q* ∈ ⋃{*state-property-decisions* σ |σ. σ ∈ σ*-set*}}› *setcompr-eq-image*)
qed
hence *consensus-value-properties-are-consistent* (⋃ {*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. $\sigma \in \sigma$*-set*})
by *simp*
qed

fun *consensus-value-property-not* :: *consensus-value-property* $\Rightarrow$ *consensus-value-property*
where
consensus-value-property-not p = (λc. ($\neg$ p c))

lemma (**in** *Protocol*) *negation-is-not-decided-by-other-validator* :
$\forall$ σ*-set*. σ*-set* $\subseteq$ Σt
$\longrightarrow$ *finite* σ*-set*
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ σ*-set*)
$\longrightarrow$ ($\forall$ σ σ' p. {σ, σ'} $\subseteq$ σ*-set* $\wedge$ $p \in$ *consensus-value-property-decisions* σ
$\longrightarrow$ *consensus-value-property-not* $p \notin$ *consensus-value-property-decisions* σ')
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix σ*-set* σ σ' p
assume σ*-set* $\subseteq$ Σt **and** *finite* σ*-set* **and** *is-faults-lt-threshold* ($\bigcup\sigma$*-set*) **and** {σ, σ'} $\subseteq$ σ*-set* $\wedge$ $p \in$ *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 *state-property-is-decided* (*naturally-corresponding-state-property* p, σ'')
using ‹{σ, σ'} $\subseteq$ σ*-set* $\wedge$ $p \in$ *consensus-value-property-decisions* σ› *consensus-value-property-decisions-def consensus-value-property-is-decided-def*
using ‹σ*-set* $\subseteq$ Σt› *forward-consistency* **by** *fastforce*
have $\sigma'' \in$ *futures* σ'
using ‹$\sigma'' \in \Sigma t$ $\wedge$ $\sigma'' \in \bigcap$ {*futures* σ | σ. $\sigma \in \sigma$*-set*}› ‹{σ, σ'} $\subseteq$ σ*-set* $\wedge$ $p \in$ *consensus-value-property-decisions* σ›
by *auto*
hence $\neg$ *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 ‹$\sigma'' \in \Sigma t$ $\wedge$ $\sigma'' \in \bigcap$ {*futures* σ | σ. $\sigma \in \sigma$*-set*}› ‹σ*-set* $\subseteq$ Σt› ‹{σ, σ'} $\subseteq$ σ*-set* $\wedge$ $p \in$ *consensus-value-property-decisions* σ› **by** *auto*
then show *consensus-value-property-not* $p \notin$ *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 Σt*-def estimates-are-non-empty futures-def* **by** *fastforce*
qed

lemma (**in** *Protocol*) *n-party-consensus-safety* :
  $\forall\ \sigma\text{-}set.\ \sigma\text{-}set \subseteq \Sigma t$
  $\longrightarrow$ *finite* σ-*set*
  $\longrightarrow$ *is-faults-lt-threshold* $(\bigcup\ \sigma\text{-}set)$
  $\longrightarrow (\forall\ \ p \in \bigcup\ \{consensus\text{-}value\text{-}property\text{-}decisions\ \sigma' \mid \sigma'.\ \sigma' \in \sigma\text{-}set\}.$
    $(\lambda c.\ (\neg\ p\ c)) \notin \bigcup\ \{consensus\text{-}value\text{-}property\text{-}decisions\ \sigma' \mid \sigma'.\ \sigma' \in \sigma\text{-}set\})$
  apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
  fix σ-*set* *p*
  assume $\sigma\text{-}set \subseteq \Sigma t$ **and** *finite* σ-*set* **and** *is-faults-lt-threshold* $(\bigcup \sigma\text{-}set)$ **and** p
$\in \bigcup\ \{consensus\text{-}value\text{-}property\text{-}decisions\ \sigma' \mid \sigma'.\ \sigma' \in \sigma\text{-}set\}$
  and $(\lambda c.\ (\neg\ p\ c)) \in \bigcup\ \{consensus\text{-}value\text{-}property\text{-}decisions\ \sigma' \mid \sigma'.\ \sigma' \in \sigma\text{-}set\}$
  hence $\exists\ \sigma.\ \sigma \in \Sigma t \wedge \sigma \in \bigcap\ \{futures\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\}$
    using *n-party-common-futures-exists* **by** *meson*
  then obtain σ'' **where** $\sigma'' \in \Sigma t \wedge \sigma'' \in \bigcap\ \{futures\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\}$ **by** *auto*
  hence *state-property-is-decided* (*naturally-corresponding-state-property* p, σ'')
   using $\langle p \in \bigcup\ \{consensus\text{-}value\text{-}property\text{-}decisions\ \sigma' \mid \sigma'.\ \sigma' \in \sigma\text{-}set\}\rangle$ *consensus-value-property-decisions-de*
consensus-value-property-is-decided-def
    using $\langle \sigma\text{-}set \subseteq \Sigma t\rangle$ *forward-consistency* **by** *fastforce*
  have *state-property-is-decided* (*naturally-corresponding-state-property* $(\lambda c.\ (\neg\ p$
$c))$, σ'')
    using $\langle(\lambda c.\ (\neg\ p\ c)) \in \bigcup\ \{consensus\text{-}value\text{-}property\text{-}decisions\ \sigma' \mid \sigma'.\ \sigma' \in$
$\sigma\text{-}set\}\rangle$ *consensus-value-property-decisions-def consensus-value-property-is-decided-def*

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

lemma (**in** *Protocol*) *two-party-consensus-safety-for-consensus-value-property* :
  $\forall\ \sigma 1\ \sigma 2.\ \sigma 1 \in \Sigma t \wedge \sigma 2 \in \Sigma t$
  $\longrightarrow$ *is-faults-lt-threshold* $(\sigma 1 \cup \sigma 2)$
  $\longrightarrow$ *consensus-value-property-is-decided* $(p, \sigma 1)$
  $\longrightarrow \neg$ *consensus-value-property-is-decided* (*consensus-value-property-not* p, $\sigma 2$)
  apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof −
  fix $\sigma 1\ \sigma 2$
  have *two-party*: $\forall\ \sigma 1\ \sigma 2.\ \{\sigma 1, \sigma 2\} \subseteq \Sigma t$
    $\longrightarrow$ *is-faults-lt-threshold* $(\bigcup\ \{\sigma 1, \sigma 2\})$
    $\longrightarrow p \in$ *consensus-value-property-decisions* $\sigma 1$
      $\longrightarrow$ *consensus-value-property-not* $p \notin$ *consensus-value-property-decisions*
$\sigma 2$
    using *negation-is-not-decided-by-other-validator*
    by (*meson finite.emptyI finite.insertI order-refl*)

assume $\sigma 1 \in \Sigma t \wedge \sigma 2 \in \Sigma t$ **and** *is-faults-lt-threshold* $(\sigma 1 \cup \sigma 2)$ **and** *consensus-value-property-is-decided* $(p, \sigma 1)$
then show $\neg$ *consensus-value-property-is-decided* (*consensus-value-property-not* $p, \sigma 2$)
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'. $\sigma' \in \sigma$*-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 Σt*-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 = $(\lambda(p, \sigma, v).\ \forall\ c \in$ *L-H-E* $\sigma\ v.\ p\ c)$

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

lemma (**in** *Protocol*) *agreeing-validators-type* :
$\forall\ \sigma \in \Sigma$. *agreeing-validators* $(p, \sigma) \subseteq V$
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* $(p, \sigma) \subseteq$ *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* $(p, \sigma) \cap$ *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 $= (\lambda(p, \sigma).\ V -$ *agreeing-validators* $(p, \sigma) -$ *equivocating-validators* $\sigma)$

lemma (**in** *Protocol*) *disagreeing-validators-type* :
$\forall\ \sigma \in \Sigma.$ *disagreeing-validators* $(p, \sigma) \subseteq V$
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* $(p, \sigma) = \{v \in V -$ *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* $(p, \sigma) \cup$ *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* $(p, \sigma), \sigma$) $\longrightarrow$ ($\forall\ c \in \varepsilon\ \sigma.\ p\ c$))

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 $p\ \sigma$ =
($\forall\ \sigma' \in \Sigma.$ *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*) ⇒ *message set*
where
later-disagreeing-messages = ($\lambda(p, m, v, \sigma).\{m' \in$ *later-from* $(m, v, \sigma).\ \neg\ p$ (*est* m')})

lemma (**in** *Protocol*) *later-disagreeing-messages-type* :
$\forall\ p\ \sigma\ v\ m.\ \sigma \in \Sigma \wedge v \in V \wedge m \in M \longrightarrow$ *later-disagreeing-messages* $(p, m, v, \sigma) \subseteq M$
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 $= (\lambda($*v-set*, p, $\sigma)$.
$(\forall\ v \in$ *v-set*. $v \in$ *observed-non-equivocating-validators* σ
$\wedge\ (\forall\ v' \in$ *v-set*.
agreeing (p, (*the-elem* (*L-H-J* σ v)), 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*) *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* σ 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 ... = $\{m'' \in \sigma.\ sender\ m'' = v \wedge justified\ m\ m''\} \cup \emptyset$
by *auto*
also have ... = $\{m'' \in \sigma.\ sender\ m'' = v \wedge justified\ m\ m''\} \cup \{m'' \in \{m'\}.\ sender\ m'' = v\}$
proof−
have $\{m'' \in \{m'\}.\ sender\ m'' = v\} = \emptyset$
using ‹$v \in V - \{sender\ m'\}$› **by** *auto*
thus *?thesis*
by *blast*
qed
also have ... = $\{m'' \in \sigma.\ sender\ m'' = v \wedge justified\ m\ m''\} \cup \{m'' \in \{m'\}.\ sender\ m'' = v \wedge justified\ m\ m''\}$
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 ‹$(\sigma, \sigma') \in minimal\text{-}transitions \wedge m \in M$› ‹$m' = the\text{-}elem\ (\sigma' - \sigma)$› *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* :
$\forall\ \sigma\ \sigma'\ m'\ v.\ (\sigma, \sigma') \in minimal\text{-}transitions$
$\longrightarrow m' = the\text{-}elem\ (\sigma' - \sigma)$
$\longrightarrow v \in V - \{sender\ m'\}$
$\longrightarrow v \in equivocating\text{-}validators\ \sigma \longleftrightarrow v \in equivocating\text{-}validators\ \sigma'$
oops

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

$\longrightarrow$ *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*
:
$\forall$ σ σ' 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, σ) = *later-disagreeing-messages* $(p, m,$
$v, \sigma')$
oops

lemma (**in** *Protocol*) *clique-not-affected-by-message-from-non-member* :
$\forall$ σ 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* :
$\forall$ σ σ' 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, σ')
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* :
$\forall\ \sigma\ \sigma'\ m'\ v.\ (\sigma, \sigma') \in minimal\text{-}transitions$
$\longrightarrow m' = the\text{-}elem\ (\sigma' - \sigma)$
$\longrightarrow v \notin equivocating\text{-}validators\ \sigma'$
$\longrightarrow m' = the\text{-}elem\ (L\text{-}H\text{-}M\ \sigma'\ v)$
oops

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

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

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

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

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

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

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

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

lemma (**in** *Protocol*) *later-disagreeing-is-monotonic*:

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

lemma (**in** *Protocol*) *empty-later-disagreeing-messages-in-new-message* :
$\forall\ \sigma\ \sigma'\ m'$ *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)), $v,\ \sigma) = \emptyset$
$\longrightarrow$ *later-disagreeing-messages* $(p,$ (*the-elem* (*L-H-M* (*justification* m') v)), $v,\ \sigma) = \emptyset$
oops

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

definition (**in** *Params*) *gt-threshold* :: (*validator set* $*$ *state*) $\Rightarrow$ *bool*
where
gt-threshold
$= (\lambda($*v-set*$,\ \sigma).($*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* :
$\forall\ \sigma$ *v-set* $p.\ \sigma \in \Sigma \wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *gt-threshold* (*v-set*, σ)
$\longrightarrow (\forall\ v \in$ *v-set*. *majority* (*v-set*, *the-elem* (*L-H-J* $\sigma\ v$)))
oops

definition (**in** *Params*) *is-clique-oracle* :: (*validator set* $*$ *state* $*$ *consensus-value-property*)
$\Rightarrow$ *bool*
where
is-clique-oracle
$= (\lambda(\textit{v-set}, \sigma, p). (\textit{is-clique}\ (\textit{v-set}, p, \sigma) \wedge \textit{gt-threshold}\ (\textit{v-set}, \sigma)))$

lemma (**in** *Protocol*) *clique-oracles-preserved-over-message-from-non-member* :
$\forall\ \sigma\ m\ \textit{v-set}\ p.\ \sigma \in \Sigma t \wedge m \in M \wedge \textit{v-set} \subseteq V$
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow$ *sender* $m \notin$ *v-set*
$\longrightarrow$ *is-clique-oracle* $(\textit{v-set}, \sigma, p)$
$\longrightarrow$ *is-clique-oracle* $(\textit{v-set}, \sigma \cup \{m\}, p)$
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*
:
$\forall\ \sigma\ m\ \textit{v-set}\ p.\ \sigma \in \Sigma t \wedge m \in M \wedge \textit{v-set} \subseteq V$
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow$ *sender* $m \in$ *v-set*
$\longrightarrow \neg$ *is-equivocating* $(\sigma \cup \{m\})$ (*sender m*)
$\longrightarrow$ *is-clique-oracle* $(\textit{v-set}, \sigma, p)$
$\longrightarrow$ *is-clique-oracle* $(\textit{v-set}, \sigma \cup \{m\}, p)$
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*
:
$\forall\ \sigma\ m\ \textit{v-set}\ p.\ \sigma \in \Sigma t \wedge m \in M \wedge \textit{v-set} \subseteq V$
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow$ *sender* $m \in$ *v-set*
$\longrightarrow$ *is-equivocating* $(\sigma \cup \{m\})$ (*sender m*)
$\longrightarrow \sigma \cup \{m\} \in \Sigma t$
$\longrightarrow$ *is-clique-oracle* $(\textit{v-set}, \sigma, p)$
$\longrightarrow$ *is-clique-oracle* $(\textit{v-set}, \sigma \cup \{m\}, p)$

sorry

lemma (**in** *Protocol*) *clique-oracles-preserved-over-immediately-next-message* :
$\forall\ \sigma\ m\ v\text{-}set\ p.\ \sigma \in \Sigma t \wedge v\text{-}set \subseteq V$
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *immediately-next-message* $(\sigma,\ m)$
$\longrightarrow \sigma \cup \{m\} \in \Sigma t$
$\longrightarrow$ *is-clique-oracle* $(v\text{-}set,\ \sigma,\ p)$
$\longrightarrow$ *is-clique-oracle* $(v\text{-}set,\ \sigma \cup \{m\},\ p)$
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* Σ*t-def insert-subset mem-Collect-eq state-is-subset-of-M*)

lemma (**in** *Protocol*) *clique-imps-everyone-agreeing* :
$\forall\ \sigma\ v\text{-}set\ p.\ \sigma \in \Sigma \wedge v\text{-}set \subseteq V$
$\longrightarrow$ *is-clique* $(v\text{-}set,\ p,\ \sigma)$
$\longrightarrow v\text{-}set \subseteq$ *agreeing-validators* $(p,\ \sigma)$
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof−
fix σ *v-set p* **assume** $\sigma \in \Sigma \wedge v\text{-}set \subseteq V$ **and** *is-clique* $(v\text{-}set,\ p,\ \sigma)$
then have *clique*: $\forall\ v \in v\text{-}set.\ v \in$ *observed-non-equivocating-validators* σ
$\wedge$ *later-disagreeing-messages* (*p*,
the-elem (*L-H-M*
(*the-elem* (*L-H-J* σ *v*)) *v*)
, *v*, σ) $= \emptyset$
by (*simp add*: *is-clique-def*)
then have *p-on-est* : $\forall\ v \in v\text{-}set.\ (\forall\ m \in \{m' \in \sigma.\ sender\ m' = v$
$\wedge$ *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 $\forall\ v \in v\text{-}set.\ v \in$ *observed-non-equivocating-validators* σ
using *clique* **by** *simp*
then have $\forall\ v \in v\text{-}set.$ *the-elem* (*L-H-J* σ *v*)
$=$ *justification* (*the-elem* (*L-H-M* σ *v*))
apply (*simp add*: *L-H-J-def*)
by (*metis* ‹$\sigma \in \Sigma \wedge v\text{-}set \subseteq V$› *empty-iff is-singleton-the-elem L-H-M-of-observed-non-equivocating-validator-singletonD singletonI the-elem-image-unique*)
then have *justified-ok*: $\forall\ v \in v\text{-}set.$ *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 ∈ Σ*t* ∧ *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 ∈ Σ*t* ∧ *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* Σ*t-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 ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› ‹*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* ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› *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 ‹$c \in \varepsilon\ \sigma$›
using *Mi.simps Σt-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*) *clique-oracle-for-all-futures* :
∀ σ *v-set p*. $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$
⟶ *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 $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$ **and** *majority-driven p* **and** *is-clique-oracle* (*v-set*, σ, *p*) **and** $\sigma' \in$ *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* :
$\forall$ σ *v-set p*. $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *finite v-set*
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *is-clique-oracle* (*v-set*, σ, *p*)
$\longrightarrow$ ($\forall$ $\sigma' \in$ *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 $\forall$ $\sigma' \in$ *futures* σ. *is-clique-oracle* (*v-set*, σ', *p*)
using *clique-oracle-for-all-futures*
by *blast*
then have $\forall$ $\sigma' \in$ *futures* σ. $\forall$ $c \in \varepsilon\ \sigma'$. *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*) ⇒ *bool*
where
agreeing = (λ(*p*, σ, *v*). ∀ *c* ∈ *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* :
$\forall\ \sigma \in \Sigma.$ *disagreeing-validators* $(p, \sigma) = \{v \in V -$ *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* $(p, \sigma) \cup$ *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*) $\Rightarrow$ *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* $(p, \sigma), \sigma) \longrightarrow (\forall\ c \in \varepsilon\ \sigma.\ p\ c)$)

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

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

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

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*) $\Rightarrow$ *bool*
where
inspector
$= (\lambda(\textit{v-set}, \sigma, p).\ \textit{v-set} \neq \emptyset \wedge$
$(\forall\ v \in \textit{v-set}.\ v \in \textit{agreeing-validators}\ (p, \sigma)$
$\wedge (\exists\ \textit{v-set}'.\ \textit{v-set}' \subseteq \textit{v-set} \wedge \textit{gt-threshold}(\textit{v-set}', \textit{the-elem}\ (\textit{L-H-J}\ \sigma\ v))$
$\wedge (\forall\ v' \in \textit{v-set}'.$
$v' \in \textit{agreeing-validators}\ (p, (\textit{the-elem}\ (\textit{L-H-J}\ \sigma\ v)))$
$\wedge\ \textit{later-disagreeing-messages}\ (p, \textit{the-elem}\ (\textit{L-H-M}\ (\textit{the-elem}\ (\textit{L-H-J}\ \sigma\ v))\ v'), v', \sigma) = \emptyset))))$

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

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

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

lemma (**in** *Protocol*) *inspector-imps-gt-threshold* :
$\forall\ \sigma\ \textit{v-set}\ p.\ \sigma \in \Sigma \wedge \textit{v-set} \subseteq V$
$\longrightarrow \textit{inspector}\ (\textit{v-set}, \sigma, p)$
$\longrightarrow \textit{gt-threshold}(\textit{v-set}, \sigma)$
apply (*rule*+)
proof −
fix σ *v-set* p
assume $\sigma \in \Sigma \wedge \textit{v-set} \subseteq V$
assume $\textit{inspector}\ (\textit{v-set}, \sigma, p)$

hence $\exists\ v \in v\text{-}set.\ \exists\ v\text{-}set'.\ v\text{-}set' \subseteq v\text{-}set \wedge gt\text{-}threshold(v\text{-}set', the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))$
apply (*simp add*: *inspector-def*)
by *blast*
hence $\exists\ v \in v\text{-}set.\ \ gt\text{-}threshold(v\text{-}set, the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))$
apply (*simp add*: *gt-threshold-def*)
using *weight-measure-subset-gte*
by (*smt* ‹$\sigma \in \Sigma \wedge v\text{-}set \subseteq V$›)
obtain v **where** $v \in v\text{-}set \wedge\ \ gt\text{-}threshold(v\text{-}set, the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))$
using ‹$\exists\, v{\in}v\text{-}set.\ gt\text{-}threshold\ (v\text{-}set, the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))$› **by** *blast*
hence $\forall\ \sigma' \in L\text{-}H\text{-}J\ \sigma\ v.\ \sigma' \subseteq \sigma$
using *L-H-J-is-subset-of-the-state* ‹$\sigma \in \Sigma \wedge v\text{-}set \subseteq V$›
by *blast*
hence *is-singleton* $(L\text{-}H\text{-}J\ \sigma\ v) \wedge (\forall\ \sigma' \in L\text{-}H\text{-}J\ \sigma\ v.\ \sigma' \subseteq \sigma)$
using *L-H-J-is-subset-of-the-state* ‹$\sigma \in \Sigma \wedge v\text{-}set \subseteq V$› *L-H-J-of-observed-non-equivocating-validator-is-singl*
‹*inspector* ($v\text{-}set$, σ, p)›
apply (*simp add*: *inspector-def agreeing-validators-def*)
using ‹$v \in v\text{-}set \wedge gt\text{-}threshold\ (v\text{-}set, the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))$› **by** *auto*
hence *the-elem* $(L\text{-}H\text{-}J\ \sigma\ v) \subseteq \sigma$
by (*metis insert-iff is-singleton-the-elem*)
then show *gt-threshold* ($v\text{-}set$, σ)
using ‹$v \in v\text{-}set \wedge gt\text{-}threshold(v\text{-}set, the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))$›
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* ‹$\sigma \in \Sigma \wedge v\text{-}set \subseteq V$› ‹*is-singleton* $(L\text{-}H\text{-}J\ \sigma\ v) \wedge (\forall\, \sigma'{\in}L\text{-}H\text{-}J\ \sigma\ v.\ \sigma' \subseteq \sigma)$› *is-singleton-the-elem singletonI subsetCE*)
qed

lemma (**in** *Protocol*) *gt-threshold-imps-estimator-agreeing* :
$\forall\ \sigma\ v\text{-}set\ p.\ \sigma \in \Sigma t \wedge v\text{-}set \subseteq V$
$\longrightarrow$ *finite v-set*
$\longrightarrow$ *majority-driven p*
$\longrightarrow v\text{-}set \subseteq agreeing\text{-}validators\ (p, \sigma)$
$\longrightarrow gt\text{-}threshold\ (v\text{-}set, \sigma)$
$\longrightarrow (\forall\ \ c \in \varepsilon\ \sigma.\ p\ c)$
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix σ *v-set p c*
assume $\sigma \in \Sigma t \wedge v\text{-}set \subseteq V$ *finite v-set majority-driven p* $v\text{-}set \subseteq agreeing\text{-}validators\ (p, \sigma)$ $gt\text{-}threshold\ (v\text{-}set, \sigma)$ $c \in \varepsilon\ \sigma$
then have *weight-measure v-set* $\leq$ *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 ‹$\sigma \in \Sigma t \wedge v\text{-}set \subseteq V$› ‹*gt-threshold* ($v\text{-}set$, σ)›

gt-threshold-def
Σ*t-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 Σ*t-def* ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› *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* Σ*t-is-subset-of-*Σ ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› *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* $(v, \sigma) =$ *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*
:
$\forall\ \sigma\ m\ v.\ \sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow v \in V - \{$*sender* $m\}$
$\longrightarrow v \in$ *equivocating-validators* $\sigma \longleftrightarrow v \in$ *equivocating-validators* $(\sigma \cup \{m\})$
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix $\sigma\ m\ v$
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* $\sigma\ v =$ *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* $\sigma) = (v \in$ *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* :
$\forall\ \sigma\ m\ v.\ \sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow v \in V - \{$*sender* $m\}$
$\longrightarrow$ *L-H-M* $\sigma\ v =$ *L-H-M* $(\sigma \cup \{m\})\ v$
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix $\sigma\ m\ v$
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* $\sigma\ v =$ *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* :
$\forall\ \sigma\ m\ v\ p.\ \sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *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* :
$\forall\ \sigma\ m\ v.\ \sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow v \in V - \{$*sender m*$\}$
$\longrightarrow$ *L-H-J* $\sigma\ v =$ *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* :
$\forall\ \sigma\ m\ m'\ v.\ \sigma \in \Sigma \wedge m \in M \wedge m' \in M \wedge v \in V$

$\longrightarrow$ *immediately-next-message* $(\sigma,\ m')$
$\longrightarrow$ $v \in V - \{$*sender* $m'\}$
$\longrightarrow$ *later-disagreeing-messages* $(p,\ m,\ v,\ \sigma) =$ *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* $(\sigma,\ m)$
$\longrightarrow$ *sender* $m \notin$ *v-set*
$\longrightarrow$ *inspector* (*v-set*, σ, p)
$\longrightarrow$ *inspector* (*v-set*, $\sigma \cup \{m\}$, p)
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix σ m *v-set* p
assume $\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$ *immediately-next-message* $(\sigma,\ m)$ *sender* $m \notin$ *v-set* *inspector* (*v-set*, σ, p)

then have $\forall$ $v \in$ *v-set*. $v \in$ *agreeing-validators* $(p,\ \sigma) \longrightarrow v \in$ *agreeing-validators* $(p,\ \sigma \cup \{m\})$
using *agreeing-status-of-non-sender-not-affected-by-minimal-transitions*
by *blast*

moreover have $\forall$ $v \in$ *v-set*.
($\forall$ *v-set*$'$. *gt-threshold*(*v-set*$'$, *the-elem* (*L-H-J* σ v)) $\longrightarrow$ *gt-threshold*(*v-set*$'$, *the-elem* (*L-H-J* ($\sigma \cup \{m\}$) v)))
using ‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› ‹*immediately-next-message* $(\sigma,\ m)$› ‹*sender* $m \notin$ *v-set*›
L-H-J-of-non-sender-not-affected-by-minimal-transitions
by *fastforce*

moreover have $\forall$ $v \in$ *v-set*.
($\forall$ *v-set*$'$. *v-set*$' \subseteq$ *v-set* $\wedge$
($\forall$ $v' \in$ *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) $= \emptyset$)
$\longrightarrow$ ($\forall$ $v' \in$ *v-set*$'$.
$v' \in$ *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', ($\sigma \cup \{m\}$)) $= \emptyset$))
apply (*rule*, *rule*, *rule*, *rule*)
proof −
fix v *v-set*$'$ v'
assume $v \in$ *v-set*
and *a1*: *v-set*$' \subseteq$ *v-set* $\wedge$ ($\forall$ $v' \in$ *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) $= \emptyset$)
and $v' \in$ *v-set'*
then have *l1*: $v' \in$ *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) $= \emptyset$
by *blast*
have $v \in$ *observed-non-equivocating-validators* σ
using ‹$v \in$ *v-set*› ‹*inspector* (*v-set*, σ, p)› *inspector-imps-everyone-observed-non-equivocating* ‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› **by** *blast*
have $v' \in$ *observed-non-equivocating-validators* (*the-elem* (*L-H-J* σ v))
using *l1* **by** (*simp add*: *agreeing-validators-def*)
then have $v' \in V - \{$*sender* $m\}$
using ‹$\sigma \in \Sigma \wedge m \in M \wedge$ *v-set* $\subseteq V$› ‹*sender* $m \notin$ *v-set*› ‹$v' \in$ *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*, $\sigma \cup \{m\}$, *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*
:
$\forall$ *v* σ $\sigma1$ $\sigma2$. $\sigma \in \Sigma \wedge \sigma1 \in \Sigma \wedge \sigma1 \subseteq \sigma \wedge \sigma2 \subseteq \sigma \wedge \sigma1 \cap \sigma2 = \emptyset$
$\longrightarrow$ ($\forall$ *m1* $\in \sigma1$. *sender m1* = *v*
$\longrightarrow$ ($\forall$ *m2* $\in \sigma2$. *sender m2* = *v* $\longrightarrow$ *m1* $\in$ *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* :
$\forall$ σ *m v*. $\sigma \in \Sigma \wedge m \in M$
$\longrightarrow$ *immediately-next-message* (σ, *m*)
$\longrightarrow$ *sender m* $\notin$ *equivocating-validators* ($\sigma \cup \{m\}$)
$\longrightarrow$ *m* = *the-elem* (*L-H-M* ($\sigma \cup \{m\}$) (*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* :
$\forall\ \sigma\ m\ v.\ \sigma \in \Sigma \wedge m \in M$
$\longrightarrow$ *immediately-next-message* $(\sigma,\ m)$
$\longrightarrow$ *sender* $m \notin$ *equivocating-validators* $(\sigma \cup \{m\})$
$\longrightarrow$ *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*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*:
$\forall\ \sigma\ m\ v.\ \sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow$ *immediately-next-message* $(\sigma,\ m)$
$\longrightarrow$ *sender* $m \notin$ *equivocating-validators* $(\sigma \cup \{m\})$
$\longrightarrow v \notin$ *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*:
$\forall\ \sigma\ m1\ m2.\ \sigma \in \Sigma \wedge \{m1,\ m2\} \subseteq \sigma$
$\longrightarrow$ *justified* $m1\ m2$
$\longrightarrow m2 \in$ *later-from* $(m1,$ *sender* $m2,\ \sigma)$
by (*simp add*: *later-from-def later-def from-sender-def*)

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

$\forall\ \sigma\ m\ v\text{-}set\ p.\ \sigma \in \Sigma \wedge m \in M \wedge v\text{-}set \subseteq V$
$\longrightarrow$ *immediately-next-message* (σ, m)
$\longrightarrow$ *sender* $m \in$ *v-set*
$\longrightarrow$ *sender* $m \notin$ *equivocating-validators* ($\sigma \cup \{m\}$)
$\longrightarrow$ *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* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \wedge m \in M$
$\longrightarrow$ *immediately-next-message* (σ, 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* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \wedge m \in M$
$\longrightarrow$ *immediately-next-message* (σ, 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* :
$\forall\ \sigma\ m\ v.\ \sigma \in \Sigma \wedge m \in M \wedge v \in V$
$\longrightarrow v \notin$ *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* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \wedge m \in M$
$\longrightarrow$ *immediately-next-message* (σ, 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* σ *m1 m2 p*. *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)
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*
:
$\forall$ σ *m v p*. $\sigma \in \Sigma$ $\wedge$ *m* $\in$ *M* $\wedge$ *v* $\in$ *V*
$\longrightarrow$ *immediately-next-message* (σ, *m*)
$\longrightarrow$ *sender m* $\notin$ *equivocating-validators* ($\sigma \cup$ {*m*})
$\longrightarrow$ *later-disagreeing-messages* (*p*, (*the-elem* (*L-H-M* (*the-elem* (*L-H-J* σ (*sender m*))) *v*)), *v*, σ) = $\emptyset$
$\longrightarrow$ *later-disagreeing-messages* (*p*, (*the-elem* (*L-H-M* (*justification m*) *v*)), *v*, σ) = $\emptyset$
apply (*simp add*: *later-disagreeing-messages-def*)
sorry

lemma (**in** *Protocol*) *inspector-preserved-over-message-from-non-equivocating-member*
:
$\forall$ σ *m v-set p*. $\sigma \in \Sigma t$ $\wedge$ *m* $\in$ *M* $\wedge$ *v-set* $\subseteq$ *V*
$\longrightarrow$ *finite v-set*
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *immediately-next-message* (σ, *m*)
$\longrightarrow$ *sender m* $\in$ *v-set*
$\longrightarrow$ *sender m* $\notin$ *equivocating-validators* ($\sigma \cup$ {*m*})
$\longrightarrow$ *inspector* (*v-set*, σ, *p*)
$\longrightarrow$ *inspector* (*v-set*, $\sigma \cup$ {*m*}, *p*)
apply (*rule*+)
proof −
fix σ *m v-set p*
assume $\sigma \in \Sigma t$ $\wedge$ *m* $\in$ *M* $\wedge$ *v-set* $\subseteq$ *V finite v-set majority-driven p immediately-next-message* (σ, *m*) *sender m* $\in$ *v-set*
sender m $\notin$ *equivocating-validators* ($\sigma \cup$ {*m*}) *inspector* (*v-set*, σ, *p*)

then have $\sigma \cup$ {*m*} $\in \Sigma t$
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* $\in$ *observed-non-equivocating-validators* ($\sigma \cup$ {*m*})
using *inspector-imps-everyone-observed-non-equivocating* ‹*inspector* (*v-set*, σ, *p*)› ‹$\sigma \in \Sigma t$ $\wedge$ *m* $\in$ *M* $\wedge$ *v-set* $\subseteq$ *V*› ‹*sender m* $\notin$ *equivocating-validators* (σ $\cup$

$\{m\})\rangle$
  apply (*simp add*: *observed-non-equivocating-validators-def observed-def*)
  by *blast*
 then have *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*)) = *justification m*
  using *new-justification-is-L-H-J-of-sender*
   $\langle\sigma \in \Sigma t \wedge m \in M \wedge$ *v-set* $\subseteq V\rangle$ $\langle$*immediately-next-message* $(\sigma, m)\rangle$ $\langle$*sender* $m \notin$ *equivocating-validators* $(\sigma \cup \{m\})\rangle$
  by (*simp add*: Σt*-def*)

 then moreover have $\forall$ $v \in$ *v-set*.
   ($\forall$ *v-set'*. *v-set'* $\subseteq$ *v-set* $\wedge$ *gt-threshold*(*v-set'*, *the-elem* (*L-H-J* σ v)) $\longrightarrow$ *gt-threshold*(*v-set'*, *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ v)))
  using $\langle\sigma \in \Sigma t \wedge m \in M \wedge$ *v-set* $\subseteq V\rangle$ $\langle$*immediately-next-message* $(\sigma, m)\rangle$ $\langle$*sender m* $\in$ *v-set*$\rangle$
   L-H-J-of-non-sender-not-affected-by-minimal-transitions
  sorry

 then moreover have $\forall$ $v \in$ *v-set*. $v \in$ *agreeing-validators* $(p, \sigma \cup \{m\})$
 proof −
  have *sender m* $\in$ *agreeing-validators* $(p, \sigma \cup \{m\})$
  proof −
   have $\forall$ *v-set'*. *v-set'* $\subseteq$ *v-set* $\longrightarrow$ *v-set'* $\subseteq$ *agreeing-validators* (*p*, *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*)))
    using *new-message-see-all-members-agreeing*
     by (*smt Protocol.new-message-see-all-members-agreeing Protocol-axioms* Σt*-is-subset-of-*Σ $\langle\sigma \in \Sigma t \wedge m \in M \wedge$ *v-set* $\subseteq V\rangle$ $\langle$*immediately-next-message* $(\sigma, m)\rangle$ $\langle$*inspector* (*v-set*, σ, *p*)$\rangle$ $\langle$*sender m* $\in$ *v-set*$\rangle$ $\langle$*sender m* $\notin$ *equivocating-validators* $(\sigma \cup \{m\})\rangle$ $\langle$*the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*)) = *justification m*$\rangle$ *subsetCE subset-trans*)
   have $\exists$ *v-set'*. *v-set'* $\subseteq$ *v-set* $\wedge$ *gt-threshold*(*v-set'*, *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*)))
    using $\langle$*inspector* (*v-set*, σ, *p*)$\rangle$
    apply (*simp add*: *inspector-def*)
     using $\langle\forall v \in$*v-set*. $\forall$ *v-set'*. *v-set'* $\subseteq$ *v-set* $\wedge$ *gt-threshold* (*v-set'*, *the-elem* (*L-H-J* σ v)) $\longrightarrow$ *gt-threshold* (*v-set'*, *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ v))$\rangle$
      $\langle$*sender m* $\in$ *v-set*$\rangle$ $\langle$*the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*)) = *justification m*$\rangle$
     by (*smt Un-insert-right* Σt*-is-subset-of-*Σ $\langle\sigma \in \Sigma t \wedge m \in M \wedge$ *v-set* $\subseteq V\rangle$ $\langle$*immediately-next-message* $(\sigma, m)\rangle$ $\langle$*inspector* (*v-set*, σ, *p*)$\rangle$ $\langle$*sender m* $\notin$ *equivocating-validators* $(\sigma \cup \{m\})\rangle$ *subsetCE subset-trans sup-bot.right-neutral*)
   then have $\exists$ *v-set'*. *v-set'* $\subseteq V \wedge$ *finite v-set'*
     $\wedge$ *v-set'* $\subseteq$ *agreeing-validators* (*p*, *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*))) $\wedge$ *gt-threshold*(*v-set'*, *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*)))
    using $\langle$*finite v-set*$\rangle$ $\langle\sigma \in \Sigma t \wedge m \in M \wedge$ *v-set* $\subseteq V\rangle$ $\langle\forall$ *v-set'*. *v-set'* $\subseteq$ *v-set* $\longrightarrow$ *v-set'* $\subseteq$ *agreeing-validators* (*p*, *the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*)))$\rangle$
    by (*meson rev-finite-subset subset-trans*)
   then have $\forall$ $c \in \varepsilon$ (*the-elem* (*L-H-J* $(\sigma \cup \{m\})$ (*sender m*))). *p c*
   using $\langle$*majority-driven p*$\rangle$ $\langle$*sender m* $\in$ *v-set*$\rangle$ *gt-threshold-imps-estimator-agreeing* $\langle\sigma \in \Sigma t \wedge m \in M \wedge$ *v-set* $\subseteq V\rangle$

‹sender m ∈ observed-non-equivocating-validators (σ ∪ {m})› ‹σ ∪ {m} ∈ Σt› ‹the-elem (L-H-J (σ ∪ {m}) (sender m)) = justification m›
past-state-of-Σt-is-Σt state-transition-is-immediately-next-message M-type
unfolding Σt-def
by (smt Σt-def Σt-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} ∈ Σt› 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*
:
$\forall\ \sigma\ m\ v\text{-}set\ p.\ \sigma \in \Sigma \wedge m \in M \wedge v\text{-}set \subseteq V$
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *immediately-next-message* $(\sigma,\ m)$
$\longrightarrow$ *sender m* $\in$ *v-set*
$\longrightarrow$ *sender m* $\in$ *equivocating-validators* $(\sigma \cup \{m\})$
$\longrightarrow \sigma \cup \{m\} \in \Sigma t$
$\longrightarrow$ *inspector* $(v\text{-}set,\ \sigma,\ p)$
$\longrightarrow$ *inspector* $(v\text{-}set,\ \sigma \cup \{m\},\ p)$

sorry

lemma (**in** *Protocol*) *inspector-preserved-over-immediately-next-message* :
$\forall\ \sigma\ m\ v\text{-}set\ p.\ \sigma \in \Sigma t \wedge v\text{-}set \subseteq V$
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *immediately-next-message* $(\sigma,\ m)$
$\longrightarrow \sigma \cup \{m\} \in \Sigma t$
$\longrightarrow$ *inspector* $(v\text{-}set,\ \sigma,\ p)$
$\longrightarrow$ *inspector* $(v\text{-}set,\ \sigma \cup \{m\},\ p)$
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-presereved-forever* :
$\forall\ \sigma\ v\text{-}set\ p.\ \sigma \in \Sigma t \wedge v\text{-}set \subseteq V$
$\longrightarrow$ *majority-driven p*
$\longrightarrow$ *inspector* $(v\text{-}set,\ \sigma,\ p)$
$\longrightarrow (\forall\ \sigma' \in$ *futures* $\sigma.$ *inspector* $(v\text{-}set,\ \sigma',\ p))$
apply (*rule*+)
proof −
fix σ *v-set p* σ'
assume $\sigma \in \Sigma t \wedge v\text{-}set \subseteq V$ **and** *majority-driven p* **and** *inspector* $(v\text{-}set,\ \sigma,\ p)$
and $\sigma' \in$ *futures* σ
then show *inspector* $(v\text{-}set,\ \sigma',\ p)$
apply (*cases* $\sigma = \sigma'$)
apply *blast*
proof −

fix σ *v-set* p σ'
assume $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$ **and** *majority-driven* p **and** *inspector* (*v-set*, σ, p) **and** $\sigma' \in$ *futures* σ **and** $\sigma \neq \sigma'$
then have $\sigma \subset \sigma'$
by (*simp add*: *futures-def psubsetI*)

then show *inspector* (*v-set*, σ', p)
using ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› ‹*majority-driven* p›
using *inspector-preserved-over-immediately-next-message state-is-finite intermediate-state-by-immediately-next-message-towards-strict-future*
sorry
qed
qed

lemma (**in** *Protocol*) *inspector-preserved-forever-by-induction* :
$\forall$ σ *v-set* p. $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *majority-driven* p
$\longrightarrow$ *inspector* (*v-set*, σ, p)
$\longrightarrow$ ($\forall$ $\sigma' \in$ *futures* σ. *inspector* (*v-set*, σ', p))
proof −
have $\forall$ n. $\forall$ σ *v-set* p. $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *majority-driven* p
$\longrightarrow$ *inspector* (*v-set*, σ, p)
$\longrightarrow$ ($\forall$ $\sigma' \in$ *futures* σ. *card* $(\sigma' - \sigma) = n \longrightarrow$ *inspector* (*v-set*, σ', p))
apply (*rule*)
proof −
fix n
show $\forall \sigma$ *v-set* p. $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V \longrightarrow$
majority-driven $p \longrightarrow$
inspector (*v-set*, σ, p) $\longrightarrow$
($\forall \sigma' \in$ *futures* σ.
card $(\sigma' - \sigma) = n \longrightarrow$
inspector (*v-set*, σ', p))
apply (*induction* n)
apply (*simp add*: *futures-def*)
using Σ*t-is-subset-of-*Σ *state-is-finite* **apply** *auto*[*1*]
apply (*rule*+)
proof −
fix n σ *v-set* p σ'
assume *a1*: $\forall \sigma$ *v-set* p.
$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V \longrightarrow$
majority-driven $p \longrightarrow$
inspector (*v-set*, σ, p) $\longrightarrow$
($\forall \sigma' \in$ *futures* σ.
card $(\sigma' - \sigma) = n \longrightarrow$
inspector (*v-set*, σ', p))
and $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$
and *majority-driven* p
and *inspector* (*v-set*, σ, p)

and $\sigma' \in$ *futures* σ
and *card* $(\sigma' - \sigma) =$ *Suc n*
then have $\sigma' \in \Sigma \wedge \sigma' \neq \emptyset$
apply (*simp add*: *futures-def*)
by (*metis* Σ*t-is-subset-of-*Σ *card-Diff-subset card-mono diff-is-0-eq' finite.emptyI nat.simps(3) subsetCE subset-empty*)
have $\sigma \subset \sigma'$
using ‹$\sigma' \in$ *futures* σ› ‹*card* $(\sigma' - \sigma) =$ *Suc n*›
apply (*simp add*: *futures-def* Σ*t-def*)
by *force*
then have $\exists\ m\ \sigma''.\ \sigma'' \in \Sigma \wedge m \in \sigma' \wedge$ *immediately-next-message* $(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\} \wedge \sigma \subseteq \sigma''$
using *intermediate-state-before-receiving-single-message* ‹$\sigma' \in \Sigma \wedge \sigma' \neq \emptyset$› ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$›
apply (*simp add*: Σ*t-def*)
by *blast*
then obtain $m\ \sigma''$ **where** $\sigma'' \in \Sigma \wedge m \in \sigma' \wedge$ *immediately-next-message* $(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\} \wedge \sigma \subseteq \sigma''$
by *auto*
then have $\sigma'' \in$ *futures* σ
using *past-state-of-*Σ*t-is-*Σ*t* ‹$\sigma' \in$ *futures* σ›
apply (*simp add*: *futures-def*)
by *blast*
have *is-singleton* $(\sigma' - \sigma'')$
using ‹$\sigma'' \in \Sigma \wedge m \in \sigma' \wedge$ *immediately-next-message* $(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\} \wedge \sigma \subseteq \sigma''$› ‹$\sigma' \in \Sigma \wedge \sigma' \neq \emptyset$›
by (*simp add*: *immediately-next-message-def insert-Diff-if*)
then have *card* $(\sigma'' - \sigma) = n$
using ‹*card* $(\sigma' - \sigma) =$ *Suc n*›
by (*smt Suc-diff-le Un-insert-right* Σ*t-is-subset-of-*Σ ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› ‹$\sigma \subset \sigma'$› ‹$\sigma'' \in \Sigma \wedge m \in \sigma' \wedge$ *immediately-next-message* $(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\} \wedge \sigma \subseteq \sigma''$› *add-left-cancel card.insert card-Diff-subset card-mono message-in-state-is-valid plus-1-eq-Suc psubsetE state-is-finite state-transition-only-made-by-immediately-next-message subsetCE sup-bot.right-neutral*)
then have *inspector* (*v-set*, σ'', *p*)
using ‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› ‹*majority-driven p*› ‹*inspector* (*v-set*, σ, *p*)› *a1*
‹$\sigma'' \in \Sigma \wedge m \in \sigma' \wedge$ *immediately-next-message* $(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\} \wedge \sigma \subseteq \sigma''$›
‹$\sigma'' \in$ *futures* σ› **by** *auto*
then show *inspector* (*v-set*, σ', *p*)
using *inspector-preserved-over-immediately-next-message*
‹$\sigma'' \in \Sigma \wedge m \in \sigma' \wedge$ *immediately-next-message* $(\sigma'', m) \wedge \sigma' = \sigma'' \cup \{m\} \wedge \sigma \subseteq \sigma''$›
‹$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$› ‹$\sigma' \in$ *futures* σ› ‹$\sigma'' \in$ *futures* σ› ‹*majority-driven p*› *futures-def*
by *auto*
qed
qed
then show *?thesis*

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-forever 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* $\Rightarrow$ *consensus-value*

fun (**in** *BlockchainParams*) *n-cestor* :: *consensus-value* $*$ *nat* $\Rightarrow$ *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* $\Rightarrow$ *consensus-value* $\Rightarrow$ *bool* (**infixl** $\downharpoonright$ *70*)
where
b1 $\downharpoonright$ *b2* = ($\exists$ *n*. $n \in \mathbb{N} \wedge$ *b1* = *n-cestor* (*b2*, *n*))

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

lemma (**in** *BlockchainParams*) *prev-membership* :
prev b $\downharpoonright$ *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* :

$b \downharpoonleft b$
apply (*simp add*: *blockchain-membership-def*)
using *Nats-0* **by** *fastforce*

definition (**in** *BlockchainParams*) *block-membership* :: *consensus-value* ⇒ *consensus-value-property*
where
block-membership $b = (\lambda b'.\ b \downharpoonleft b')$

lemma (**in** *BlockchainParams*) *also-agreeing-on-ancestors* :
$b' \downharpoonleft b \Longrightarrow$ *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* ⇒ *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* $(b', \sigma).\ \ b' \downharpoonleft b$
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* : $\forall\ b\ b'\ b''.\ \{b, b', b''\} \subseteq C \longrightarrow b' \downharpoonleft b \land b'' \downharpoonleft b \longrightarrow (b' \downharpoonleft b'' \lor b'' \downharpoonleft b')$
and *children-conflicting* : $\forall\ \sigma \in \Sigma.\ \forall\ b\ b1\ b2.\ \{b, b1, b2\} \subseteq C \land \{b1, b2\} \subseteq$ *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* $\downharpoonleft b$ *prev genesis* = *genesis*

lemma (**in** *Blockchain*) *children-type* :
$\forall\ b\ \sigma.\ b \in C \land \sigma \in \Sigma \longrightarrow$ *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* :
$\forall\ b1\ b2\ \sigma.\ \{b1,\ b2\} \subseteq C \wedge \sigma \in \Sigma$
$\longrightarrow$ *block-conflicting* ($b1$, $b2$)
$\longrightarrow$ *consensus-value-property-is-decided* (*block-membership* $b1$, σ)
$\longrightarrow$ *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 $b1\ b2\ \sigma$
assume $b1 \in C \wedge b2 \in C \wedge \sigma \in \Sigma$ **and** *block-conflicting* ($b1$, $b2$) **and** $\forall\,\sigma{\in}futures\ \sigma.\ \forall\, b'{\in}\varepsilon\ \sigma.\ b1\ \downharpoonleft\ b'$
show $\forall\,\sigma{\in}futures\ \sigma.\ \forall\, c{\in}\varepsilon\ \sigma.\ \neg\ b2\ \downharpoonleft\ c$
proof (*rule ccontr*)
assume $\neg\ (\forall\,\sigma{\in}futures\ \sigma.\ \forall\, c{\in}\varepsilon\ \sigma.\ \neg\ b2\ \downharpoonleft\ c)$
hence $\exists\ \sigma\ {\in}futures\ \sigma.\ \exists\ c \in \varepsilon\ \sigma.\ b2\ \downharpoonleft\ c$
by *blast*
hence $\exists\ \sigma\ {\in}futures\ \sigma.\ \exists\ c \in \varepsilon\ \sigma.\ b2\ \downharpoonleft\ c \wedge b1\ \downharpoonleft\ c$
using ‹$\forall\,\sigma{\in}futures\ \sigma.\ \forall\, b'{\in}\varepsilon\ \sigma.\ b1\ \downharpoonleft\ b'$› **by** *simp*
hence $b1\ \downharpoonleft\ b2 \vee b2\ \downharpoonleft\ b1$
using *blockchain-type*
apply (*simp*)
using Σ*t-is-subset-of-*Σ ‹$b1 \in C \wedge b2 \in C \wedge \sigma \in \Sigma$› *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* :
$\forall\ \sigma\text{-}set.\ \sigma\text{-}set \subseteq \Sigma t$
$\longrightarrow$ *finite* σ*-set*
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup\ \sigma$*-set*)
$\longrightarrow$ ($\forall\ \sigma\ \sigma'\ b1\ b2.\ \{\sigma,\ \sigma'\} \subseteq \sigma\text{-}set \wedge \{b1,\ b2\} \subseteq C \wedge$ *block-conflicting* ($b1$, $b2$) $\wedge$ *block-membership* $b1 \in$ *consensus-value-property-decisions* σ
$\longrightarrow$ *block-membership* $b2 \notin$ *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* ⊆ Σ*t* **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* ⊆ Σ*t*› ‹*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 Σ*t-is-subset-of-*Σ ‹*σ-set* ⊆ Σ*t*› ‹{*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* ⊆ Σ*t*› ‹*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*} ⊆ Σ*t*
⟶ *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*} ⊆ Σ*t* **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*

$(\sigma 1 \cup \sigma 2)$› ‹$\{\sigma 1, \sigma 2\} \subseteq \Sigma t$› **by** *blast*
  have *block-membership b2* $\in$ *consensus-value-property-decisions* $\sigma 2$
    using ‹*block-membership b2* $\in$ *consensus-value-property-decisions* $\sigma 2$›
    by (*simp add*: *consensus-value-property-decisions-def*)
  have $\sigma 2 \in \Sigma t \wedge \{b2, b1\} \subseteq C \wedge$ *block-conflicting* $(b2, b1)$
    using ‹$\{\sigma 1, \sigma 2\} \subseteq \Sigma t$› ‹$\{b1, b2\} \subseteq C \wedge$ *block-conflicting* $(b1, b2)$› **by** (*simp add*: *block-conflicting-def*)
 hence *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership b1*), $\sigma 2$)
   using *conflicting-blocks-imps-conflicting-decision* ‹*block-membership b2* $\in$ *consensus-value-property-decision* $\sigma 2$›
    using Σ*t-is-subset-of-*Σ *consensus-value-property-decisions-def* **by** *auto*
  then show *False*
      using ‹$\neg$ *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership b1*), $\sigma 2$)› **by** *blast*
 qed

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

lemma (**in** *Blockchain*) *unfolding-agreeing-on-block-membership* :
 $\forall \sigma \in \Sigma.$ *agreeing-validators* (*block-membership b*, σ) = $\{v \in V. \exists b' \in$ *L-H-E* σ $v.$ $b \downharpoonleft b'\}$
proof −
  have $\forall v\ \sigma.\ v \in V \wedge \sigma \in \Sigma \longrightarrow v \notin$ *equivocating-validators* σ
        $\longrightarrow$ ($v \in$ *observed* $\sigma \wedge (\forall x \in$ *L-M* σ $v.$ $b \downharpoonleft$ *est* x)) = ($v \in$ *observed* $\sigma \wedge (\exists x \in$*L-M* σ $v.$ $b \downharpoonleft$ *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 $\forall v\ \sigma.\ v \in V \wedge \sigma \in \Sigma \longrightarrow v \notin$ *equivocating-validators* σ
        $\longrightarrow$ ($v \in V \wedge (\exists x \in$*L-M* σ $v.$ $b \downharpoonleft$ *est* x)) = ($v \in$ *observed* $\sigma \wedge (\exists x \in$*L-M* σ $v.$ $b \downharpoonleft$ *est* x))
    apply (*simp add*: *observed-def L-M-def from-sender-def*)
    by *auto*
  ultimately have $\forall v\ \sigma.\ v \in V \wedge \sigma \in \Sigma \longrightarrow v \notin$ *equivocating-validators* σ
        $\longrightarrow$ ($v \in V \wedge (\exists x \in$*L-M* σ $v.$ $b \downharpoonleft$ *est* x)) = ($v \in$ *observed* $\sigma \wedge (\forall x \in$ *L-M* σ $v.$ $b \downharpoonleft$ *est* x))
    by *blast*
  then have $\forall v\ \sigma.\ v \in V \wedge \sigma \in \Sigma$
        $\longrightarrow$ ($v \notin$ *equivocating-validators* $\sigma \longrightarrow v \in V \wedge (\exists x \in$*L-M* σ $v.$ $b \downharpoonleft$ *est* x)) = ($v \notin$ *equivocating-validators* $\sigma \longrightarrow v \in$ *observed* $\sigma \wedge (\forall x \in$ *L-M* σ $v.$ $b \downharpoonleft$

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 ‹$\forall\ v\ \sigma.\ v \in V \wedge \sigma \in \Sigma$
   $\longrightarrow (v \notin$ *equivocating-validators* $\sigma \longrightarrow v \in V \wedge (\exists\ x \in$*L-M* $\sigma\ v.\ b \downharpoonright$ *est* $x)) = (v \notin$ *equivocating-validators* $\sigma \longrightarrow v \in$ *observed* $\sigma \wedge (\forall\ x \in$ *L-M* $\sigma\ v.\ b \downharpoonright$ *est* $x))$›
  observed-type-for-state
  by *blast*
qed

definition (**in** *BlockchainParams*) *score-magnitude* :: *state* $\Rightarrow$ *consensus-value rel*
 where
  score-magnitude $\sigma = \{(b1,\ b2).\ \{b1,\ b2\} \subseteq C \wedge$ *score* σ $b1 \leq$ *score* σ $b2\}$

lemma (**in** *Blockchain*) *transitivity-of-score-magnitude* :
 $\forall\ \sigma \in \Sigma.$ *trans* (*score-magnitude* σ)
 by (*simp add*: *trans-def score-magnitude-def*)

lemma (**in** *Blockchain*) *reflexivity-of-score-magnitude* :
 $\forall\ \sigma \in \Sigma.$ *refl-on* C (*score-magnitude* σ)
 apply (*simp add*: *refl-on-def score-magnitude-def*)
 by *auto*

lemma (**in** *Blockchain*) *score-magnitude-is-preorder* :
 $\forall\ \sigma \in \Sigma.$ *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* :
 $\forall\ \sigma \in \Sigma.$ *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* $\Rightarrow$ *consensus-value* $\Rightarrow$ *consensus-value rel*
 where
  score-magnitude-children σ $b = \{(b1,\ b2).\ \{b1,\ b2\} \subseteq$ *children* $(b, \sigma) \wedge$ *score* σ $b1 \leq$ *score* σ $b2\}$

lemma (**in** *Blockchain*) *transitivity-of-score-magnitude-children* :
 $\forall\ \sigma \in \Sigma.\ \forall\ b \in C.$ *trans* (*score-magnitude-children* σ b)
 by (*simp add*: *trans-def score-magnitude-children-def*)

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

lemma (**in** *Blockchain*) *best-children-finite* :
$\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow$ *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* :
$\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow$ *children* (b, σ) $\neq \emptyset \longrightarrow$ *best-children* (b, σ) $\in$ *Pow* $C - \{\emptyset\}$
proof −
have $\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow$ *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* ⇒ *state-property*
where
best-child b = ($\lambda\sigma$. *b* ∈ *best-children* (*prev b*, σ))

function (**in** *BlockchainParams*) *GHOST* :: (*consensus-value set* ∗ *state*) ⇒ *consensus-value set*
where
GHOST (*b-set*, σ) =
($\bigcup$ *b* ∈ {*b* ∈ *b-set*. *children* (*b*, σ) ≠ ∅}. *GHOST* (*best-children* (*b*, σ), σ))
∪ {*b* ∈ *b-set*. *children* (*b*, σ) = ∅}
by *auto*

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

lemma (**in** *Blockchain*) *GHOST-type* :
∀ σ *b-set*. $\sigma \in \Sigma$ ∧ *b-set* ⊆ *C* ⟶ *GHOST* (*b-set*, σ) ⊆ *C*
proof −

have ∀ σ *b-set*. $\sigma \in \Sigma$ ∧ *b-set* ⊆ *C* ⟶ (∃ *b-set'*. *b-set'* ⊆ *C* ∧ *GHOST* (*b-set*, σ) = {*b* ∈ *b-set'*. *children* (*b*, σ) = ∅})
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, σ)
$\longrightarrow$ *agreeing-validators* (*block-membership* $b1$, σ) $\subseteq$ *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, σ)
$\longrightarrow$ ($\forall\ v \in$ *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 $\forall\ \sigma \in \Sigma.\ \forall\ b\ b1\ b2.\ \{b,\ b1,\ b2\} \subseteq C \wedge \{b1,\ b2\} \subseteq$ *children* (b, σ)
$\longrightarrow$ ($\forall\ v \in$ *agreeing-validators* (*block-membership* $b1$, σ). $\exists\ c \in$*L-H-E* σ v. $\neg$ *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*) ‹$\forall\sigma\in\Sigma.\ \forall\ b\ b1\ b2.\ \{b,\ b1,\ b2\} \subseteq C \wedge \{b1,\ b2\} \subseteq$ *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)› *insert-subset subsetI*)
qed

lemma (**in** *Blockchain*) *agreeing-validators-on-sistor-blocks-are-not-more-than-disagreeing* :
$\forall\ \sigma \in \Sigma.\ \forall\ b\ b1\ b2.\ \{b,\ b1,\ b2\} \subseteq C \wedge \{b1,\ b2\} \subseteq$ *children* (b, σ)
$\longrightarrow$ *weight-measure* (*agreeing-validators* (*block-membership* $b1$, σ)) $\leq$ *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* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in C.$ *children* $(b, \sigma) = \emptyset \wedge$
($\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in$ *best-children* (*prev* b', σ))
$\longrightarrow b \in$ *GHOST* ($\{$*genesis*$\}$, σ)
apply *auto*
oops

lemma (**in** *Blockchain*) *best-child-at-all-earlier-height-imps-GHOST-heads-or-decendant* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in C.$
($\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in$ *best-children* (*prev* b', σ))
$\longrightarrow$ ($\forall\ b'' \in$ *GHOST* ($\{$*genesis*$\}$, σ). $b \downharpoonleft b''$)

```
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* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in est$ ‘$\sigma.\ \forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in est$ ‘σ
sorry

lemma (**in** *TFG*) *block-membership-is-max-driven* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in est$ ‘σ. *max-driven-for-future* (*block-membership* b) σ
apply (*simp add*: *max-driven-for-future-def*)
proof −
have $\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b,\ b'\} \subseteq C \wedge b' \downharpoonleft b$
$\longrightarrow$ *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 $\forall\ b' \in C.\ b' \downharpoonright$ *est* $m \longrightarrow$ *weight-measure* (*agreeing-validators* (*block-membership* b', σ')) $>$ *weight-measure* (*disagreeing-validators* (*block-membership* b', σ'))
using ‹$\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b, b'\} \subseteq C \wedge b' \downharpoonright b$
$\longrightarrow$ *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 $\forall\ b' \in C.\ b' \downharpoonright$ *est* $m \longrightarrow b' \in$ *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* ‹$\sigma \subseteq \sigma'$› ‹$\sigma' \in \Sigma$› ‹$m \in \sigma$› *contra-subsetD image-eqI observed-block-is-children-of-prev-block*)

using *M-type Params.message-in-state-is-valid* ‹$\sigma \in \Sigma$›
using *agreeing-validators-on-sistor-blocks-are-not-more-than-disagreeing prev-type*
‹$\forall\ b' \in C.\ b' \downharpoonright$ *est* $m \longrightarrow$ *weight-measure* (*agreeing-validators* (*block-membership* b', σ')) $>$ *weight-measure* (*disagreeing-validators* (*block-membership* b', σ'))›
by (*smt* ‹$\sigma' \in \Sigma$› *agreeing-validators-weight-combined children-type contra-subsetD empty-subsetI insert-absorb2 insert-subset*)
have $c \in$ *GHOST* ($\{$*genesis*$\}$, σ') $\cup$ ($\bigcup\ b \in$ *GHOST* ($\{$*genesis*$\}$, σ'). *children* (b, σ'))
using *ghost-estimator* ‹$c \in \varepsilon\ \sigma'$›
unfolding *GHOST-heads-or-children-def*
by *blast*
have $\forall\ b'' \in$ *GHOST* ($\{$*genesis*$\}$, σ'). *est* $m \downharpoonright b''$
using *best-child-at-all-earlier-height-imps-GHOST-heads-or-decendant* ‹$\forall\ b' \in C.\ b' \downharpoonright$ *est* $m \longrightarrow b' \in$ *best-children* (*prev* b', σ')›
‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$› ‹*est* $m \in C$› **by** *blast*
then show *block-membership* (*est* m) c
unfolding *block-membership-def*
using ‹$c \in$ *GHOST* ($\{$*genesis*$\}$, σ') $\cup$ ($\bigcup\ b \in$ *GHOST* ($\{$*genesis*$\}$, σ'). *children* (b, σ'))›
transitivity-of-blockchain-membership children-membership
by *blast*
qed
qed

end