Minimal CBC Casper Isabelle/HOL proofs

LayerX

June 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

$maximum\text{-}on\ A\ r\ x = (x \in A \wedge upper\text{-}bound\text{-}on\ A\ r\ x)$

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

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

lemma *maximal-and-maximum-coincide-for-strict-linear-order* :
$strict\text{-}linear\text{-}order\text{-}on\ A\ r \Longrightarrow maximal\text{-}on\ A\ r\ x = maximum\text{-}on\ A\ r\ x$
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* :
$strict\text{-}partial\text{-}order\ r \longrightarrow finite\ A \longrightarrow A \neq \emptyset \longrightarrow (\exists\ x.\ maximal\text{-}on\ A\ r\ x)$
proof −
have $\bigwedge n.\ strict\text{-}partial\text{-}order\ r \Longrightarrow (\forall\ A.\ Suc\ n = card\ A \longrightarrow finite\ A \longrightarrow A \neq \emptyset \longrightarrow (\exists\ x.\ maximal\text{-}on\ A\ r\ x))$
proof −
assume $strict\text{-}partial\text{-}order\ r$
then have $(\forall a.\ (a,\ a) \notin r)$
by (*simp add*: *strict-partial-order-def irrefl-def*)
fix n
show $\forall\ A.\ Suc\ n = card\ A \longrightarrow finite\ A \longrightarrow A \neq \emptyset \longrightarrow (\exists\ x.\ maximal\text{-}on\ A\ r\ x)$
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* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow is\text{-}singleton\ \{x.\ maximum\text{-}on\ A\ r\ x\})$
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow \neg\ is\text{-}singleton\ \{x.\ maximum\text{-}on\ A\ r\ x\}$
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* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow is\text{-}singleton\ \{x.\ maximum\text{-}on\ A\ r\ x\})$› *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* $\longrightarrow A \neq \emptyset \longrightarrow is\text{-}singleton\ \{x.\ maximum\text{-}on\ A\ r\ x\}$
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* :
$\forall\ \sigma \in \Sigma.\ (\exists\ n \in \mathbb{N}.\ \sigma \in Pow\ (Mi\ (V,\ C,\ \varepsilon)\ (n - 1)) \land (\forall\ m \in \sigma.\ justification\ m \subseteq \sigma))$
apply (*simp add*: Σ*-def*)

apply *auto*
proof –
fix $y :: nat$ **and** $\sigma :: message\ set$
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 $\exists n.\ n \in \mathbb{N} \land \sigma \subseteq Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
using *a2* **by** (*metis* (*no-types*) *Nats-1 Nats-add diff-Suc-1 plus-1-eq-Suc*)
then show $\exists n\in\mathbb{N}.\ \sigma \subseteq \{m.\ est\ m \in C \land sender\ m \in V \land justification\ m \in \Sigma i\ (V,\ C,\ \varepsilon)\ (n - Suc\ 0) \land est\ m \in \varepsilon\ (justification\ m)\}$

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.*Σ*i-monotonic* **by** *fastforce*
qed

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

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

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

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

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

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

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

end

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

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

then have $\{m'\} \in$ *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, ε) (*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*(σ, σ')
  $\longrightarrow \sigma' - \sigma \subseteq M$
  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*) $\Rightarrow$ *bool*
where
equivocation =
(λ(*m1*, *m2*). *sender m1* = *sender m2* $\wedge$ *m1* $\neq$ *m2* $\wedge$ $\neg$ (*justified m1 m2*) $\wedge$ $\neg$ (*justified m2 m1*))

definition *is-equivocating* :: *state* $\Rightarrow$ *validator* $\Rightarrow$ *bool*
where
is-equivocating σ *v* = ($\exists$ *m1* $\in$ σ. $\exists$ *m2* $\in$ σ. *equivocation* (*m1*, *m2*) $\wedge$ *sender m1* = *v*)

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

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

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

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

lemma (**in** *Protocol*) *equivocating-validators-is-equivalent-to-paper* :
$\forall$ $\sigma \in \Sigma$. *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* :
$\forall$ σ σ' *v*. *is-future-state* (σ, σ') $\wedge$ *v* $\in V$
$\longrightarrow$ *v* $\in$ *equivocating-validators* σ
$\longrightarrow$ *v* $\in$ *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* :
$\forall$ σ *m*. $\sigma \in \Sigma \wedge m \in M$
$\longrightarrow$ $\neg$ *is-equivocating* ($\sigma \cup \{m\}$) (*sender m*)
$\longrightarrow$ *equivocating-validators* σ = *equivocating-validators* ($\sigma \cup \{m\}$)
apply (*simp add*: *equivocating-validators-def is-equivocating-def observed-def equivocation-def*)

by *auto*

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

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

lemma (**in** *Params*) *weight-measure-subset-minus* :
finite A ⟹ *finite B* ⟹ $A \subseteq B$
⟹ *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* $A \land$ *finite* B
using *V-type finite-subset* ‹$B \subseteq V$› ‹$A \subset B$› **by** *auto*
have $B - A \neq \emptyset \land B - A \subseteq V$
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* $A \land$ *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* :
$\forall\ \sigma\ \sigma'.\ \sigma \in \Sigma \land \sigma' \in \Sigma \land$ *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*

type-synonym *state-property* = *state* ⇒ *bool*

type-synonym *consensus-value-property* = *consensus-value* ⇒ *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*} ⊆ *M* ∧ *justified m1 m2*}

lemma (**in** *Protocol*) *transitivity-of-justifications* :
trans message-justification
apply (*simp add*: *trans-def message-justification-def justified-def*)
by (*meson Params.M-type Params.state-is-in-pow-Mi Protocol-axioms contra-subsetD*)

lemma (**in** *Protocol*) *irreflexivity-of-justifications* :
irrefl message-justification
apply (*simp add*: *irrefl-def message-justification-def justified-def*)
apply (*simp add*: *M-def*)
apply *auto*
proof −
fix *n m*
assume *est m* ∈ *C*
assume *sender m* ∈ *V*
assume *justification m* ∈ Σi (*V*, *C*, ε) *n*
assume *est m* ∈ ε (*justification m*)

assume $m \in$ *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* ∈ *C*› ‹*est m* ∈ ε (*justification m*)› ‹*justification m* ∈ Σ*i* (*V*, *C*, ε) *n*› ‹*m* ∈ *justification m*› ‹*sender m* ∈ *V*› *add.right-neutral add-Suc-right diff-is-0-eq′ diff-le-self diff-zero mem-Collect-eq not-gr0 subsetCE*)
then have *justification m* $\in \Sigma i\ (V,\ C,\ \varepsilon)\ (n - 1)$
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* ∈ *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* ∈ *justification m*› **by** *blast*
qed

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

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

lemma (**in** *Protocol*) *monotonicity-of-justifications* :
∀ *m m′ σ*. *m* ∈ *M* ∧ σ ∈ Σ ∧ *justified m′ m* ⟶ *justification m′* ⊆ *justification m*
apply *simp*
by (*meson M-type justified-def message-in-state-is-valid state-is-in-pow-Mi*)

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

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

lemma (**in** *Protocol*) *only-valid-message-is-justified* :
$\forall\ m \in M.\ \forall\ m'.$ *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* $\Sigma i.simps(1)$ 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*) ⇒ *message set*
where
later = (λ(*m*, σ). {*m*′ ∈ σ. *justified m m*′})

lemma (**in** *Protocol*) *later-type* :
∀ σ *m*. σ ∈ *Pow M* ∧ *m* ∈ *M* ⟶ *later* (*m*, σ) ⊆ *M*
apply (*simp add*: *later-def*)
by *auto*

lemma (**in** *Protocol*) *later-type-for-state* :
∀ σ *m*. σ ∈ Σ ∧ *m* ∈ *M* ⟶ *later* (*m*, σ) ⊆ *M*
apply (*simp add*: *later-def*)
using *state-is-subset-of-M* **by** *auto*

definition *from-sender* :: (*validator* ∗ *message set*) ⇒ *message set*
where
from-sender = (λ(*v*, σ). {*m* ∈ σ. *sender m* = *v*})

lemma (**in** *Protocol*) *from-sender-type* :
∀ σ *v*. σ ∈ *Pow M* ∧ *v* ∈ *V* ⟶ *from-sender* (*v*, σ) ∈ *Pow M*
apply (*simp add*: *from-sender-def*)
by *auto*

lemma (**in** *Protocol*) *from-sender-type-for-state* :
∀ σ *v*. σ ∈ Σ ∧ *v* ∈ *V* ⟶ *from-sender* (*v*, σ) ⊆ *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* :
∀ σ *v*. σ ∈ Σ ∧ *v* ∈ *observed* σ ⟶ *from-sender* (*v*, σ) ≠ ∅
apply (*simp add*: *observed-def from-sender-def*)
by *auto*

lemma (**in** *Protocol*) *messages-from-validator-is-finite* :
∀ σ *v*. σ ∈ Σ ∧ *v* ∈ *V*σ ⟶ *finite* (*from-sender* (*v*, σ))
by (*simp add*: *from-sender-def state-is-finite*)

definition *from-group* :: (*validator set* ∗ *message set*) ⇒ *state*
where
from-group = (λ(*v-set*, σ). {*m* ∈ σ. *sender m* ∈ *v-set*})

lemma (**in** *Protocol*) *from-group-type* :
∀ σ *v*. σ ∈ *Pow M* ∧ *v-set* ⊆ *V* ⟶ *from-group* (*v-set*, σ) ∈ *Pow M*
apply (*simp add*: *from-group-def*)

by *auto*

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

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

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

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

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

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

lemma (**in** *Protocol*) *L-M-type-for-state* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow L\text{-}M\ \sigma\ v \subseteq M$
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* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \wedge v \notin observed\ \sigma \longrightarrow L\text{-}M\ \sigma\ v = \emptyset$
by (*simp add*: *L-M-def observed-def later-def from-sender-def*)

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

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, σ) $\longrightarrow$ *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, σ) $\longrightarrow m1 = m2 \vee$ *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 later-def message-justification-def maximal-on-def*)
using *from-sender-type-for-state* **apply** *auto*
apply (*metis* (*no-types*, *lifting*) *IntI empty-iff from-sender-def mem-Collect-eq prod.simps*(*2*))
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 $\sigma\ v = \{$*est* $m \mid m.\ m \in$ *L-M* $\sigma\ v\}$

lemma (**in** *Protocol*) *L-E-type* :

$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *L-E* $\sigma\ v \subseteq C$
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* $\sigma\ v = \emptyset$
using *L-E-def L-M-from-non-observed-validator-is-empty* **by** *auto*

definition *L-H-M* :: *state* $\Rightarrow$ *validator* $\Rightarrow$ *message set*
where
L-H-M $\sigma\ v$ = (*if* $v \in$ *equivocating-validators* σ *then* $\emptyset$ *else L-M* $\sigma\ v$)

lemma (**in** *Protocol*) *L-H-M-type* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *L-H-M* $\sigma\ v \subseteq M$
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* $\sigma\ v$)
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*
$\sigma\ v$)) $= v$
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* $\sigma\ v$)
$\in \sigma$
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

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

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

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

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

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

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

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

lemma (**in** *Protocol*) *L-H-J-of-observed-non-equivocating-validator-is-singleton* :
$\forall\ \sigma \in \Sigma.\ v \in observed\text{-}non\text{-}equivocating\text{-}validators\ \sigma$
$\longrightarrow is\text{-}singleton\ (L\text{-}H\text{-}J\ \sigma\ v)$
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* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow (\forall\ \sigma' \in L\text{-}H\text{-}J\ \sigma\ v.\ \sigma' \subset \sigma)$

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 = $\{(\sigma 1, \sigma 2).\ \{\sigma 1, \sigma 2\} \subseteq \Sigma \wedge$ *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 (**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* $(\sigma,$

σ') $\wedge$ $\sigma \neq \sigma'$
$\wedge$ ($\nexists$ σ''. $\sigma'' \in \Sigma$ $\wedge$ *is-future-state* (σ, σ'') $\wedge$ *is-future-state* (σ'', σ') $\wedge$ $\sigma \neq \sigma''$ $\wedge$ $\sigma'' \neq \sigma'$)}

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

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

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

have *justification* $m \subseteq \sigma$
using *immediately-next-message-def*
by (*metis* (*no-types*, *lifting*) ‹*immediately-next-message* (σ, m)› *case-prod-conv*)
hence *justification* $m \subseteq \sigma \cup \{m\}$
by *blast*
moreover have $\bigwedge m'$. *finite* $\sigma \wedge m' \in \sigma \Longrightarrow$ *justification* $m' \subseteq \sigma$
using ‹$\sigma \in \Sigma i$ (V, C, ε) 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,ε) n)
using ‹$m \in Mi$ (V, C, ε) n› **by** *auto*
moreover have $\sigma \in$ *Pow* (*Mi* (V,C,ε) ($n-1$))
using ‹$\sigma \in \Sigma i$ (V, C, ε) n› *si* **by** *auto*
hence $\sigma \in$ *Pow* (*Mi* (V,C,ε) 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,ε) 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* Σ*i-monotonic* Σ*i-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 Σ*i-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 σ 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 $\langle$*immediately-next-message* $(\sigma,\ m)\rangle$ **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*) *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 $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - (\sigma \cap \sigma').\ immediately\text{-}next\text{-}message(\sigma \cap \sigma', m))$
by (*metis Int-subset-iff psubsetI strict-subset-of-state-have-immediately-next-messages subsetI*)
then have $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - (\sigma \cap \sigma').\ immediately\text{-}next\text{-}message(\sigma', m))$
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 $\{the\text{-}elem\ (\sigma - \sigma')\} \cup \sigma' \in \Sigma$
using $\langle\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)\rangle$ $\langle\sigma \in \Sigma\rangle$ $\langle\sigma' \in \Sigma\rangle$
by (*metis Un-commute* $\langle\neg\ \sigma \subseteq \sigma' \wedge Suc\ 0 = card\ (\sigma - \sigma')\rangle$ *is-singleton-the-elem singletonD*)
then show $\sigma \cup \sigma' \in \Sigma$
by (*metis Un-Diff-cancel2* $\langle is\text{-}singleton\ (\sigma - \sigma')\rangle$ *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 $\langle\neg\ \sigma \subseteq \sigma' \wedge Suc\ (Suc\ n) = card\ (\sigma - \sigma')\rangle$
by (*metis Diff-eq-empty-iff Diff-insert Un-insert-right* $\langle\sigma \in \Sigma\rangle$

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

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

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

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

lemma (**in** *Protocol*) *minimal-transition-implies-recieving-single-message* :
$\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow$ *is-singleton* $(\sigma' - \sigma)$
proof (*rule ccontr*)
assume $\neg\ (\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow$ *is-singleton* $(\sigma' - \sigma))$
then have $\exists\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$
by *blast*
have $\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow$
$(\nexists\ \sigma''.\ \sigma'' \in \Sigma \wedge$ *is-future-state* $(\sigma, \sigma'') \wedge$ *is-future-state* $(\sigma'', \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma')$
by (*simp add*: *minimal-transitions-def*)
have $\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$
$\longrightarrow (\exists\ m1\ m2.\ \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$ *immediately-next-message* $(\sigma, m1))$
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ \sigma'$
assume $(\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$
then have $\sigma' - \sigma \neq \emptyset$
apply (*simp add*: *minimal-transitions-def*)
by *blast*
have $\sigma' \in \Sigma \wedge \sigma \in \Sigma \wedge$ *is-future-state* (σ, σ')
using ‹$(\sigma, \sigma') \in$ *minimal-transitions* $\wedge \neg$ *is-singleton* $(\sigma' - \sigma)$›
by (*simp add*: *minimal-transitions-def* Σ*t-def*)

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

qed
then show *False*
using ‹$\forall \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow (\nexists \sigma''.\ \sigma'' \in \Sigma \wedge$ *is-future-state* $(\sigma, \sigma'') \wedge$ *is-future-state* $(\sigma'', \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma')$› ‹$\neg (\forall \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow$ *is-singleton* $(\sigma' - \sigma))$› **by** *blast*
qed

lemma (**in** *Protocol*) *minimal-transitions-reconstruction* :
$\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow \sigma \cup \{$*the-elem* $(\sigma' - \sigma)\} = \sigma'$
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ \sigma'$
assume $(\sigma, \sigma') \in$ *minimal-transitions*
then have *is-singleton* $(\sigma' - \sigma)$
using *minimal-transitions-def minimal-transition-implies-recieving-single-message* **by** *auto*
then have $\sigma \subseteq \sigma'$
using ‹$(\sigma, \sigma') \in$ *minimal-transitions*› *minimal-transitions-def* **by** *auto*
then show $\sigma \cup \{$*the-elem* $(\sigma' - \sigma)\} = \sigma'$
by (*metis Diff-partition* ‹*is-singleton* $(\sigma' - \sigma)$› *is-singleton-the-elem*)
qed

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

oops

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

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

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

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

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

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

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

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

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

lemma (**in** *Protocol*) *backword-consistency* :
  $\forall\ \sigma'\ \sigma.\ \sigma' \in \Sigma t \wedge \sigma \in \Sigma t$
  $\longrightarrow \sigma' \in$ *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* :
  $\forall\ \sigma 1\ \sigma 2.\ \sigma 1 \in \Sigma t \wedge \sigma 2 \in \Sigma t$
  $\longrightarrow$ *is-faults-lt-threshold* $(\sigma 1 \cup \sigma 2)$
  $\longrightarrow \neg$(*state-property-is-decided* $(p, \sigma 1) \wedge$ *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 ⇒ state-property set
  where
    state-property-decisions σ = {p. state-property-is-decided (p, σ)}


theorem (in Protocol) n-party-safety-for-state-properties :
  ∀ σ-set. σ-set ⊆ Σt
  ⟶ finite σ-set
  ⟶ is-faults-lt-threshold (⋃ σ-set)
  ⟶ state-properties-are-consistent (⋃ {state-property-decisions σ | σ. σ ∈ σ-set})
  apply rule+
proof−
  fix σ-set
  assume σ-set: σ-set ⊆ Σt
  and finite σ-set
  and is-faults-lt-threshold (⋃ σ-set)
  hence ∃σ∈Σt. σ ∈ ⋂ {futures σ | σ. σ ∈ σ-set}
    using n-party-common-futures-exists by simp
  hence ∃σ∈Σt. ∀s∈σ-set. σ ∈ futures s
    by blast
  hence ∃σ∈Σt. (∀s∈σ-set. σ ∈ futures s) ∧ (∀s∈σ-set. σ ∈ futures s ⟶ (∀p.
state-property-is-decided (p,s) ⟶ state-property-is-decided (p,σ)))
    by (simp add: subset-eq state-property-is-decided-def futures-def)
 hence ∃σ∈Σt. ∀s∈σ-set. (∀p. state-property-is-decided (p,s) ⟶ state-property-is-decided
(p,σ))
    by blast
 hence ∃σ∈Σt. ∀s∈σ-set. (∀p ∈ state-property-decisions s. state-property-is-decided
(p,σ))
    by (simp add: state-property-decisions-def)
 hence ∃σ∈Σt. ∀p∈⋃{state-property-decisions σ | σ. σ ∈ σ-set}. state-property-is-decided
(p,σ)
  proof−
   obtain σ where σ ∈ Σt ∀s∈σ-set. (∀p ∈ state-property-decisions s. state-property-is-decided
(p,σ))
     using ‹∃σ∈Σt. ∀s∈σ-set. ∀p∈state-property-decisions s. state-property-is-decided
(p, σ)› by blast
   have ∀p∈⋃{state-property-decisions σ | σ. σ ∈ σ-set}. state-property-is-decided
(p,σ)
      using ‹∀s∈σ-set. ∀p∈state-property-decisions s. state-property-is-decided (p,
σ)› by fastforce
    thus ?thesis
      using ‹σ ∈ Σt› by blast
  qed
  hence ∃σ∈Σt. ∀p∈⋃{state-property-decisions σ | σ. σ ∈ σ-set}. ∀σ'∈futures
σ. p σ'
   by (simp add: state-property-decisions-def futures-def state-property-is-decided-def)
 show state-properties-are-consistent (⋃{state-property-decisions σ |σ. σ ∈ σ-set})
    unfolding state-properties-are-consistent-def
```

by (*metis* (*mono-tags*, *lifting*) Σt-*def* ‹$\exists\sigma\in\Sigma t.\ \forall p\in\bigcup\{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$-*set*$\}.\ \forall\sigma'\in$*futures* $\sigma.\ p\ \sigma'$› *mem-Collect-eq monotonic-futures order-refl*)
qed

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

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

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

$\longrightarrow (\exists\ c \in C.\ \forall\ q' \in q\text{-}set.\ q'\ c)$
using *is-valid-estimator-def* ε*-type* **by** *fastforce*
ultimately show
state-properties-are-consistent $\{$*naturally-corresponding-state-property q* $|q.\ q \in$ *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* $\Rightarrow$ *consensus-value-property set*
where
consensus-value-property-decisions $\sigma = \{q.$ *consensus-value-property-is-decided* $(q, \sigma)\}$

theorem (**in** *Protocol*) *n-party-safety-for-consensus-value-properties* :
$\forall\ \sigma$*-set*. σ*-set* $\subseteq \Sigma t$
$\longrightarrow$ *finite* σ*-set*
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup\ \sigma$*-set*)
$\longrightarrow$ *consensus-value-properties-are-consistent* ($\bigcup$ $\{$*consensus-value-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}$)
apply (*rule*, *rule*, *rule*, *rule*)
proof −
fix σ*-set*
assume σ*-set* $\subseteq \Sigma t$
and *finite* σ*-set*
and *is-faults-lt-threshold* ($\bigcup\ \sigma$*-set*)
hence *state-properties-are-consistent* ($\bigcup$ $\{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}$)
using ‹σ*-set* $\subseteq \Sigma t$› *n-party-safety-for-state-properties* **by** *auto*
hence *state-properties-are-consistent* $\{p \in \bigcup$ $\{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}$. $\exists\ q.\ p =$ *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* $\mid$ q. *naturally-corresponding-state-property q* $\in \bigcup$ $\{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}\}$
by (*smt Collect-cong*)

hence *consensus-value-properties-are-consistent* $\{q.$ *naturally-corresponding-state-property* $q \in \bigcup$ $\{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}\}$
using *naturally-corresponding-consistency*
proof −
show *?thesis*
by (*metis* (*no-types*) *Setcompr-eq-image* ‹$\forall$ *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*› ‹*state-properties-are-consistent* $\{$*naturally-corresponding-state-property* $q \mid q.$ *naturally-corresponding-state-property* $q \in \bigcup\{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}\}$› *setcompr-eq-image*)
qed
hence *consensus-value-properties-are-consistent* ($\bigcup$ $\{$*consensus-value-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-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* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}$)
by *simp*
qed

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

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

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

end
theory *SafetyOracle*

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
apply (*simp add*: *agreeing-validators-def*)
by *blast*

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, σ) $\cup$ *equivocating-validators* σ) $=$ *weight-measure* (*agreeing-validators* (p, σ)) $+$ *equivocation-fault-weight* σ
unfolding *equivocation-fault-weight-def*
using *agreeing-validators-are-not-equivocating weight-measure-disjoint-plus agreeing-validators-finite equivocating-validators-is-finite*
by *simp*

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

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

definition (**in** *Params*) *majority* :: (*validator set* $*$ *state*) $\Rightarrow$ *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, σ), σ) $\longrightarrow$ ($\forall\ c \in \varepsilon\ \sigma.\ p\ c$))

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

definition (**in** *Protocol*) *max-driven-for-future* :: *consensus-value-property* $\Rightarrow$ *state* $\Rightarrow$ *bool*
where
max-driven-for-future p σ $=$
($\forall\ \sigma' \in \Sigma$. *is-future-state* ($\sigma$, σ')

$\longrightarrow$ *weight-measure* (*agreeing-validators* (p, σ')) > *weight-measure* (*disagreeing-validators* (p, σ')) $\longrightarrow$ ($\forall$ $c \in \varepsilon$ σ'. 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 σ 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*

definition *is-clique* :: (*validator set* $*$ *consensus-value-property* $*$ *state*) $\Rightarrow$ *bool*
where
is-clique = (λ(*v-set*, p, σ).
($\forall$ $v \in$ *v-set*. $v \in$ *observed-non-equivocating-validators* σ
$\wedge$ ($\forall$ $v' \in$ *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* :
$\forall$ σ v σ'. $v \in V \wedge \{\sigma, \sigma'\} \subseteq \Sigma \wedge$ *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 σ. *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* $(m,\ v,\ \sigma')$
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* $m\ m''\}$
apply (*simp add*: *later-from-def from-sender-def later-def*)
by *auto*
also have $\ldots = \{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\} \cup \emptyset$
by *auto*
also have $\ldots = \{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\} \cup \{m'' \in \{m'\}.$
sender $m'' = v\}$
proof−
have $\{m'' \in \{m'\}.$ *sender* $m'' = v\} = \emptyset$
using ‹$v \in V - \{$*sender* $m'\}$› **by** *auto*
thus *?thesis*
by *blast*
qed
also have $\ldots = \{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\} \cup \{m'' \in \{m'\}.$
sender $m'' = v \wedge$ *justified* $m\ m''\}$
proof−
have *sender* $m' = v \Longrightarrow$ *justified* $m\ m'$
using ‹$v \in V - \{$*sender* $m'\}$› **by** *auto*
thus *?thesis*
by *blast*
qed
also have $\ldots = \{m'' \in \sigma \cup \{m'\}.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\}$
by *auto*
also have $\ldots = \{m'' \in \sigma'.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\}$
proof −
have $\sigma' = \sigma \cup \{m'\}$
using ‹$(\sigma,\ \sigma') \in$ *minimal-transitions* $\wedge\ m \in M$› ‹$m' =$ *the-elem* $(\sigma' - \sigma)$›
minimal-transitions-reconstruction **by** *auto*
then show *?thesis*
by *auto*
qed
then have $\ldots =$ *later-from* (m,v,σ')
apply (*simp add*: *later-from-def from-sender-def later-def*)
by *auto*
then show *later-from* $(m,\ v,\ \sigma) =$ *later-from* $(m,\ v,\ \sigma')$
using ‹$\{m'' \in \sigma \cup \{m'\}.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\} = \{m'' \in \sigma'.$ *sender*

$m'' = v \wedge$ *justified* m $m''\}\rangle$ *calculation* **by** *auto*
qed

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

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

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

lemma (**in** *Protocol*) *later-disagreeing-of-non-sender-not-affected-by-minimal-transitions*
:
$\forall$ 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* $(p, m, v, \sigma) =$ *later-disagreeing-messages* $(p, m,$
$v, \sigma')$
oops

lemma (**in** *Protocol*) *clique-not-affected-by-minimal-transitions-outside-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*, p, σ')
oops

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

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

lemma (**in** *Protocol*) *message-between-minimal-transition-is-latest-message* :
$\forall\ \sigma\ \sigma'\ m'\ v.\ (\sigma, \sigma') \in minimal\text{-}transitions$
$\longrightarrow m' = the\text{-}elem\ (\sigma' - \sigma)$
$\longrightarrow v \notin equivocating\text{-}validators\ \sigma'$
$\longrightarrow m' = the\text{-}elem\ (L\text{-}H\text{-}M\ \sigma'\ v)$
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* :
$\forall\ \sigma\ \sigma'\ m'\ v\text{-}set.\ (\sigma, \sigma') \in minimal\text{-}transitions \land v\text{-}set \subseteq V$
$\longrightarrow m' = the\text{-}elem\ (\sigma' - \sigma)$
$\longrightarrow is\text{-}clique\ (v\text{-}set, p, \sigma) \land sender\ m' \in v\text{-}set \land sender\ m' \notin equivocating\text{-}validators$
σ'
$\longrightarrow v\text{-}set \subseteq agreeing\text{-}validators\ (p, justification\ m')$
oops

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

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

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

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

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

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

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

lemma (**in** *Protocol*) *clique-not-affected-by-minimal-transitions-outside-clique* :
$\forall\ \sigma\ \sigma'\ m'\ v\text{-}set\ p.\ (\sigma,\ \sigma') \in minimal\text{-}transitions \wedge v\text{-}set \subseteq V$
$\longrightarrow majority\text{-}driven\ p$
$\longrightarrow m' = the\text{-}elem\ (\sigma' - \sigma)$
$\longrightarrow is\text{-}clique\ (v\text{-}set,\ p,\ \sigma) \wedge sender\ m' \in v\text{-}set \wedge sender\ m' \notin equivocating\text{-}validators\ \sigma'$
$\wedge (\forall\ v \in v\text{-}set.\ majority\ (v\text{-}set,\ the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v)))$
$\longrightarrow is\text{-}clique\ (v\text{-}set,\ p,\ \sigma')$
oops

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

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

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

lemma (**in** *Protocol*) *clique-oracles-preserved-over-minimal-transitions-from-validators-not-in-clique* :
$\forall\ \sigma\ \sigma'\ m'\ \textit{v-set}\ p.\ (\sigma,\ \sigma') \in \textit{minimal-transitions} \wedge \textit{v-set} \subseteq V$
$\longrightarrow \textit{majority-driven}\ p$
$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow \textit{sender}\ m' \notin \textit{v-set} - \textit{equivocating-validators}\ \sigma$
$\wedge\ \textit{is-clique-oracle}\ (\textit{v-set},\ \sigma,\ p)$
$\longrightarrow \textit{is-clique-oracle}\ (\textit{v-set},\ \sigma',\ p)$
oops

lemma (**in** *Protocol*) *clique-oracles-preserved-over-minimal-transitions-from-non-equivocating-validator* :
$\forall\ \sigma\ \sigma'\ m'\ \textit{v-set}\ p.\ (\sigma,\ \sigma') \in \textit{minimal-transitions} \wedge \textit{v-set} \subseteq V$
$\longrightarrow \textit{majority-driven}\ p$
$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow \textit{sender}\ m' \in \textit{v-set} - \textit{equivocating-validators}\ \sigma \wedge \textit{sender}\ m' \notin \textit{equivocating-validators}\ \sigma'$
$\wedge\ \textit{is-clique-oracle}\ (\textit{v-set},\ \sigma,\ p)$
$\longrightarrow \textit{is-clique-oracle}\ (\textit{v-set},\ \sigma',\ p)$
oops

lemma (**in** *Protocol*) *clique-oracles-preserved-over-minimal-transitions-from-equivocating-validator* :
$\forall\ \sigma\ \sigma'\ m'\ \textit{v-set}\ p.\ (\sigma,\ \sigma') \in \textit{minimal-transitions} \wedge \textit{v-set} \subseteq V$
$\longrightarrow \textit{majority-driven}\ p$
$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow \textit{sender}\ m' \in \textit{v-set} - \textit{equivocating-validators}\ \sigma \wedge \textit{sender}\ m' \in \textit{equivocating-validators}\ \sigma'$
$\wedge\ \textit{is-clique-oracle}\ (\textit{v-set},\ \sigma,\ p)$
$\longrightarrow \textit{is-clique-oracle}\ (\textit{v-set},\ \sigma',\ p)$
oops

lemma (**in** *Protocol*) *clique-oracles-preserved-over-minimal-transitions* :
$\forall\ \sigma\ \sigma'\ m'\ v\text{-}set\ p.\ (\sigma,\ \sigma') \in minimal\text{-}transitions \wedge v\text{-}set \subseteq V$
$\longrightarrow majority\text{-}driven\ p$
$\longrightarrow m' = the\text{-}elem\ (\sigma' - \sigma)$
$\longrightarrow is\text{-}clique\text{-}oracle\ (v\text{-}set,\ \sigma,\ p)$
$\longrightarrow is\text{-}clique\text{-}oracle\ (v\text{-}set,\ \sigma',\ p)$
sorry

lemma (**in** *Protocol*) *clique-oracles-preserved-over-nice-message* :
$\forall\ \sigma\ m'\ v\text{-}set\ p.\ \sigma \in \Sigma t \wedge v\text{-}set \subseteq V$
$\longrightarrow majority\text{-}driven\ p$
$\longrightarrow \sigma \cup \{m'\} \in \Sigma t$
$\longrightarrow is\text{-}clique\text{-}oracle\ (v\text{-}set,\ \sigma,\ p)$
$\longrightarrow is\text{-}clique\text{-}oracle\ (v\text{-}set,\ \sigma \cup \{m'\},\ p)$
sorry

lemma (**in** *Protocol*) *clique-imps-everyone-agreeing* :
$\forall\ \sigma\ v\text{-}set\ p.\ \sigma \in \Sigma \wedge v\text{-}set \subseteq V$
$\longrightarrow is\text{-}clique\ (v\text{-}set,\ p,\ \sigma)$
$\longrightarrow v\text{-}set \subseteq agreeing\text{-}validators\ (p,\ \sigma)$
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof−
fix σ *v-set* p **assume** $\sigma \in \Sigma \wedge v\text{-}set \subseteq V$ **and** $is\text{-}clique\ (v\text{-}set,\ p,\ \sigma)$
then have *clique*: $\forall\ v \in v\text{-}set.\ v \in observed\text{-}non\text{-}equivocating\text{-}validators\ \sigma$
$\wedge\ later\text{-}disagreeing\text{-}messages\ (p,$
$the\text{-}elem\ (L\text{-}H\text{-}M$
$(the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))\ v)$
$,\ v,\ \sigma) = \emptyset$
by (*simp add*: *is-clique-def*)
then have *p-on-est* : $\forall\ v \in v\text{-}set.\ (\forall\ m \in \{m' \in \sigma.\ sender\ m' = v$
$\wedge\ justified\ (the\text{-}elem\ (L\text{-}H\text{-}M$
$(the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))\ v))$
$m'\}.$
$p(est\ m))$
by (*simp add*: *later-disagreeing-messages-def later-from-def later-def from-sender-def*)
have $\forall\ v \in v\text{-}set.\ v \in observed\text{-}non\text{-}equivocating\text{-}validators\ \sigma$
using *clique* **by** *simp*
then have $\forall\ v \in v\text{-}set.\ the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v)$
$=\ justification\ (the\text{-}elem\ (L\text{-}H\text{-}M\ \sigma\ v))$
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*: $\forall\ v \in v\text{-}set.\ justified\ (the\text{-}elem\ (L\text{-}H\text{-}M$
$(the\text{-}elem\ (L\text{-}H\text{-}J\ \sigma\ v))\ v))$

$(\textit{the-elem}\ (\textit{L-H-M}\ \sigma\ v))$
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* $\langle\forall\, v{\in}\textit{v-set}.\ v \in \textit{observed-non-equivocating-validators}\ \sigma\rangle$ $\langle\sigma \in \Sigma \wedge \textit{v-set} \subseteq V\rangle$ $\langle\textit{is-clique}\ (\textit{v-set}, p, \sigma)\rangle$ *empty-subsetI insert-subset is-singleton-the-elem justified-def observed-non-equivocating-validators-def state-is-subset-of-M subsetCE*)
have *sender-ok*: $\forall\ v \in \textit{v-set}.\ \textit{sender}\ (\textit{the-elem}\ (\textit{L-H-M}\ \sigma\ v)) = v$
using $\langle\forall\ v \in \textit{v-set}.\ v \in \textit{observed-non-equivocating-validators}\ \sigma\rangle$ *sender-of-L-H-M*
using $\langle\sigma \in \Sigma \wedge \textit{v-set} \subseteq V\rangle$ **by** *blast*
have $\forall\ v \in \textit{v-set}.\ \ \textit{the-elem}\ (\textit{L-H-M}\ \sigma\ v) \in \sigma$
using $\langle\forall\ v \in \textit{v-set}.\ v \in \textit{observed-non-equivocating-validators}\ \sigma\rangle$ *L-H-M-is-in-the-state*
using $\langle\sigma \in \Sigma \wedge \textit{v-set} \subseteq V\rangle$ **by** *blast*
then have $\forall\ v \in \textit{v-set}.\ p\ (\textit{est}\ (\textit{the-elem}\ (\textit{L-H-M}\ \sigma\ v)))$
using *p-on-est sender-ok justified-ok*
by *blast*
then have $\forall\ v \in \textit{v-set}.\ p\ (\textit{the-elem}\ (\textit{L-H-E}\ \sigma\ v))$
apply (*simp add*: *L-H-E-def*)
by (*metis* (*no-types*, *lifting*) $\langle\forall\, v{\in}\textit{v-set}.\ v \in \textit{observed-non-equivocating-validators}\ \sigma\rangle$ $\langle\sigma \in \Sigma \wedge \textit{v-set} \subseteq V\rangle$ *empty-iff is-singleton-the-elem L-H-M-of-observed-non-equivocating-validator-is-singleto singletonD singletonI the-elem-image-unique*)
then show $\textit{v-set} \subseteq \textit{agreeing-validators}\ (p, \sigma)$
unfolding *agreeing-validators-def agreeing-def*
by (*smt* $\langle\forall\, v{\in}\textit{v-set}.\ v \in \textit{observed-non-equivocating-validators}\ \sigma\rangle$ $\langle\sigma \in \Sigma \wedge \textit{v-set} \subseteq V\rangle$ *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* :
$\forall\ \sigma\ \textit{v-set}\ p.\ \sigma \in \Sigma t \wedge \textit{v-set} \subseteq V$
$\longrightarrow$ *finite v-set*
$\longrightarrow$ *majority-driven p*
$\longrightarrow \textit{is-clique}\ (\textit{v-set} - \textit{equivocating-validators}\ \sigma,\ p,\ \sigma) \wedge \textit{gt-threshold}\ (\textit{v-set} - \textit{equivocating-validators}\ \sigma,\ \sigma)$
$\longrightarrow (\forall\ c \in \varepsilon\ \sigma.\ p\ c)$
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix σ *v-set p c*
assume $\sigma \in \Sigma t \wedge \textit{v-set} \subseteq V$
and *finite v-set*
and *majority-driven p*
and $\textit{is-clique}\ (\textit{v-set} - \textit{equivocating-validators}\ \sigma,\ p,\ \sigma) \wedge \textit{gt-threshold}\ (\textit{v-set} - \textit{equivocating-validators}\ \sigma,\ \sigma)$
and $c \in \varepsilon\ \sigma$
then have $\textit{v-set} - \textit{equivocating-validators}\ \sigma \subseteq \textit{agreeing-validators}\ (p, \sigma)$
using *clique-imps-everyone-agreeing*
by (*meson Diff-subset* Σt*-is-subset-of-*Σ *subsetCE subset-trans*)
then have $\textit{weight-measure}\ (\textit{v-set} - \textit{equivocating-validators}\ \sigma) \leq \textit{weight-measure}\ (\textit{agreeing-validators}\ (p, \sigma))$

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) $\wedge$ *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* $\sigma \subseteq V$
by (*simp add*: *Diff-subset*)
ultimately have (*weight-measure* V) *div 2* $\geq$ (*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) $\leq$ *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 ‹$c \in \varepsilon$ σ›
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* :
$\forall$ σ *v-set* p. $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *majority-driven* p
$\longrightarrow$ *is-clique-oracle* (*v-set*, σ, p)
$\longrightarrow$ ($\forall$ $\sigma' \in$ *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-minimal-transitions*
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')
using *clique-oracle-for-all-futures threshold-sized-clique-imps-estimator-agreeing*
apply (*simp add*: *is-clique-oracle-def naturally-corresponding-state-property-def*)
by (*metis* (*mono-tags*, *lifting*) *futures-def mem-Collect-eq*)

end
theory *TFGCasper*

imports *Main HOL.Real CBCCasper LatestMessage SafetyOracle 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** $\downharpoonleft$ *70*)
where
$b1 \downharpoonleft b2 = (\exists\ n.\ n \in \mathbb{N} \wedge b1 =$ *n-cestor* $(b2, n))$

notation (*ASCII*)

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

lemma (**in** *BlockchainParams*) *prev-membership* :
prev b ⇂ *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* $\Rightarrow$ *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* $\Rightarrow$ *consensus-value set*
where
children $= (\lambda(b, \sigma).\ \{b' \in est\ `\sigma.\ b = prev\ b'\})$

lemma (**in** *BlockchainParams*) *observed-block-is-children-of-prev-block* :
$\forall\ b \in est\ `\sigma.\ b \in children\ (prev\ b, \sigma)$
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* : $\forall\ b\ b'\ b''.\ \{b, b', b''\} \subseteq C \longrightarrow b' \downharpoonleft b \wedge b'' \downharpoonleft b \longrightarrow (b' \downharpoonleft b'' \vee b'' \downharpoonleft b')$
and *children-conflicting* : $\forall\ \sigma \in \Sigma.\ \forall\ b\ b1\ b2.\ \{b, b1, b2\} \subseteq C \wedge \{b1, b2\} \subseteq children\ (b, \sigma) \longrightarrow block\text{-}conflicting\ (b1, b2)$

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

lemma (**in** *Blockchain*) *children-finite* :
$\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow\ finite\ (children\ (b,\ \sigma))$
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 Σt*-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* $(\bigcup\ \sigma\text{-}set)$

$\longrightarrow (\forall\ \sigma\ \sigma'\ b1\ b2.\ \{\sigma, \sigma'\} \subseteq \sigma\text{-}set \wedge \{b1, b2\} \subseteq C \wedge block\text{-}conflicting\ (b1, b2)$
$\wedge\ block\text{-}membership\ b1 \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma$
$\longrightarrow block\text{-}membership\ b2 \notin consensus\text{-}value\text{-}property\text{-}decisions\ \sigma')$
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix $\sigma\text{-}set\ \sigma\ \sigma'\ b1\ b2$
assume $\sigma\text{-}set \subseteq \Sigma t$ **and** *finite* $\sigma\text{-}set$ **and** *is-faults-lt-threshold* $(\bigcup \sigma\text{-}set)$
and $\{\sigma, \sigma'\} \subseteq \sigma\text{-}set \wedge \{b1, b2\} \subseteq C \wedge block\text{-}conflicting\ (b1, b2) \wedge block\text{-}membership$
$b1 \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma$
and $block\text{-}membership\ b2 \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma'$
hence $\neg$ *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership*
b1), σ')
using *negation-is-not-decided-by-other-validator* ‹$\sigma\text{-}set \subseteq \Sigma t$› ‹*finite* $\sigma\text{-}set$›
‹*is-faults-lt-threshold* $(\bigcup \sigma\text{-}set)$› **apply** (*simp add*: *consensus-value-property-decisions-def*)

using ‹$\{\sigma, \sigma'\} \subseteq \sigma\text{-}set \wedge \{b1, b2\} \subseteq C \wedge block\text{-}conflicting\ (b1, b2) \wedge$
$block\text{-}membership\ b1 \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma$› **by** *auto*
have $\{b1, b2\} \subseteq C \wedge \sigma \in \Sigma \wedge block\text{-}conflicting\ (b1, b2)$
using Σt*-is-subset-of-*Σ ‹$\sigma\text{-}set \subseteq \Sigma t$› ‹$\{\sigma, \sigma'\} \subseteq \sigma\text{-}set \wedge \{b1, b2\} \subseteq C \wedge$
$block\text{-}conflicting\ (b1, b2) \wedge block\text{-}membership\ b1 \in consensus\text{-}value\text{-}property\text{-}decisions$
σ› **by** *auto*
hence *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership*
b1), σ')
using ‹$block\text{-}membership\ b2 \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma'$› *conflicting-blocks-imps-conflicting-dec*
apply (*simp add*: *consensus-value-property-decisions-def*)
by (*metis* ‹$\sigma\text{-}set \subseteq \Sigma t$› ‹*finite* $\sigma\text{-}set$› ‹*is-faults-lt-threshold* $(\bigcup \sigma\text{-}set)$› ‹$\{\sigma,$
$\sigma'\} \subseteq \sigma\text{-}set \wedge \{b1, b2\} \subseteq C \wedge block\text{-}conflicting\ (b1, b2) \wedge block\text{-}membership\ b1$
$\in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma$› *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 ‹$\neg$ *consensus-value-property-is-decided* (*consensus-value-property-not*
(*block-membership b1*), σ')› **by** *blast*
qed

theorem (**in** *Blockchain*) *no-decision-on-conflicting-blocks* :
$\forall\ \sigma 1\ \sigma 2.\ \{\sigma 1, \sigma 2\} \subseteq \Sigma t$
$\longrightarrow is\text{-}faults\text{-}lt\text{-}threshold\ (\sigma 1 \cup \sigma 2)$
$\longrightarrow (\forall\ b1\ b2.\ \{b1, b2\} \subseteq C \wedge block\text{-}conflicting\ (b1, b2)$
$\longrightarrow block\text{-}membership\ b1 \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma 1$
$\longrightarrow block\text{-}membership\ b2 \notin consensus\text{-}value\text{-}property\text{-}decisions\ \sigma 2)$
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix $\sigma 1\ \sigma 2\ b1\ b2$
assume $\{\sigma 1, \sigma 2\} \subseteq \Sigma t$ **and** *is-faults-lt-threshold* $(\sigma 1 \cup \sigma 2)$ **and** $\{b1, b2\} \subseteq C$
$\wedge\ block\text{-}conflicting\ (b1, b2)$
and $block\text{-}membership\ b1 \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma 1$
and $block\text{-}membership\ b2 \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma 2$

hence *consensus-value-property-is-decided* (*block-membership b1*, σ*1*)
by (*simp add*: *consensus-value-property-decisions-def*)
hence $\neg$ *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership b1*), σ*2*)
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* σ*2*
using ‹*block-membership b2* $\in$ *consensus-value-property-decisions* σ*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*), σ*2*)
using *conflicting-blocks-imps-conflicting-decision* ‹*block-membership b2* $\in$ *consensus-value-property-decision* σ*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*), σ*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* $\sigma\ v.\ b \downharpoonleft b'\}$
proof –
have $\forall\ v\ \sigma.\ v \in V \wedge \sigma \in \Sigma \longrightarrow v \notin$ *equivocating-validators* σ
$\longrightarrow$ ($v \in$ *observed* σ $\wedge$ ($\forall\ x \in$ *L-M* $\sigma\ v.\ b \downharpoonleft$ *est x*)) = ($v \in$ *observed* σ $\wedge$ ($\exists\ x \in$*L-M* $\sigma\ v.\ b \downharpoonleft$ *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* $\sigma\ v.\ b \downharpoonleft$ *est x*)) = ($v \in$ *observed* σ $\wedge$ ($\exists\ x \in$*L-M* $\sigma\ v.\ b \downharpoonleft$ *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* $\sigma\ v.\ b \downharpoonleft$ *est x*)) = ($v \in$ *observed* σ $\wedge$ ($\forall\ x \in$

L-M σ v. b ⌊ est x))
    by *blast*
  then have ∀ *v σ. v* ∈ *V* ∧ *σ* ∈ Σ
        ⟶ (*v* ∉ *equivocating-validators σ* ⟶ *v* ∈ *V* ∧ (∃ *x* ∈*L-M σ v. b* ⌊ *est x*)) = (*v* ∉ *equivocating-validators σ* ⟶ *v* ∈ *observed σ* ∧ (∀ *x* ∈ *L-M σ v. b* ⌊ *est x*))
    by *blast*
  show *?thesis*
   apply (*simp add*: *agreeing-validators-def agreeing-def observed-non-equivocating-validators-def L-H-E-def L-H-M-def block-membership-def*)
    using ‹∀ *v σ. v* ∈ *V* ∧ *σ* ∈ Σ
        ⟶ (*v* ∉ *equivocating-validators σ* ⟶ *v* ∈ *V* ∧ (∃ *x* ∈*L-M σ v. b* ⌊ *est x*)) = (*v* ∉ *equivocating-validators σ* ⟶ *v* ∈ *observed σ* ∧ (∀ *x* ∈ *L-M σ v. b* ⌊ *est x*))›
    observed-type-for-state
    by *blast*
qed

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

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

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

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

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

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

lemma (**in** *Blockchain*) *transitivity-of-score-magnitude-children* :

$\forall\ \sigma \in \Sigma.\ \forall\ b \in C.\ trans\ (score\text{-}magnitude\text{-}children\ \sigma\ b)$
by (*simp add*: *trans-def score-magnitude-children-def*)

lemma (**in** *Blockchain*) *reflexivity-of-score-magnitude-children* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in C.\ refl\text{-}on\ (children\ (b, \sigma))\ (score\text{-}magnitude\text{-}children\ \sigma\ b)$
apply (*simp add*: *refl-on-def score-magnitude-children-def*)
by *blast*

lemma (**in** *Blockchain*) *score-magnitude-children-is-preorder* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in C.\ preorder\text{-}on\ (children\ (b, \sigma))\ (score\text{-}magnitude\text{-}children\ \sigma\ b)$
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* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in C.\ Relation.total\text{-}on\ (children\ (b, \sigma))\ (score\text{-}magnitude\text{-}children\ \sigma\ b)$
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
$best\text{-}children = (\lambda\ (b, \sigma).\ \{b' \in C.\ is\text{-}arg\text{-}max\ (score\ \sigma)\ (\lambda b'.\ b' \in children\ (b, \sigma))\ b'\})$

lemma (**in** *Blockchain*) *best-children-type* :
$\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow best\text{-}children\ (b, \sigma) \subseteq C$
apply (*simp add*: *is-arg-max-def best-children-def*)
by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq subsetI*)

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

lemma (**in** *Blockchain*) *best-children-existence* :
$\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow children\ (b, \sigma) \neq \emptyset \longrightarrow best\text{-}children\ (b, \sigma) \in Pow\ C - \{\emptyset\}$
proof −
have $\forall\ b\ \sigma.\ b \in C \wedge \sigma \in \Sigma \longrightarrow children\ (b, \sigma) \neq \emptyset$
$\longrightarrow (\exists\ b'.\ maximum\text{-}on\text{-}non\text{-}strict\ (children\ (b, \sigma))\ (score\text{-}magnitude\text{-}children\ \sigma\ b)\ b')$
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 = ($\lambda\sigma$. $b \in$ *best-children* (*prev b*, σ))

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

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

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

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

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

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

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

lemma (**in** *Blockchain*) *agreeing-validators-on-sistor-blocks-are-disagreeing* :
  $\forall\ \sigma \in \Sigma.\ \forall\ b\ b1\ b2.\ \{b, b1, b2\} \subseteq C \wedge \{b1, b2\} \subseteq$ *children* (b, 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*
:
  $\forall\ \sigma \in \Sigma.\ \forall\ b \in C.$
    $(\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in \textit{best-children}\ (\textit{prev}\ b',\ \sigma))$
    $\longrightarrow (\forall\ b'' \in \textit{GHOST}\ (\{\textit{genesis}\},\ \sigma).\ b \downharpoonleft b'')$
proof −
  have $\bigwedge n.\ \forall\ \sigma \in \Sigma.\ \forall\ b \in C.\ \textit{genesis} = \textit{n-cestor}\ (b,\ n) \wedge$
    $(\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in \textit{best-children}\ (\textit{prev}\ b',\ \sigma))$
    $\longrightarrow (\forall\ b'' \in \textit{GHOST}\ (\{\textit{genesis}\},\ \sigma).\ b \downharpoonleft b'')$
  proof −
    fix n
    show $\forall \sigma{\in}\Sigma.\ \forall b{\in}C.\ \textit{genesis} = \textit{n-cestor}\ (b,\ n) \wedge$
      $(\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in \textit{best-children}\ (\textit{prev}\ b',\ \sigma)) \longrightarrow$
      $(\forall b''{\in}\textit{GHOST}\ (\{\textit{genesis}\},\ \sigma).\ b \downharpoonleft b'')$
      apply (*induction n*)
      using *genesis-type GHOST-type*
      apply (*metis contra-subsetD empty-subsetI insert-subset n-cestor.simps*(*1*))
    proof −
      fix n
      assume $\forall \sigma{\in}\Sigma.\ \forall b{\in}C.\ \textit{genesis} = \textit{n-cestor}\ (b,\ n) \wedge$
        $(\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in \textit{best-children}\ (\textit{prev}\ b',\ \sigma)) \longrightarrow$
        $(\forall b''{\in}\textit{GHOST}\ (\{\textit{genesis}\},\ \sigma).\ b \downharpoonleft b'')$
      show $\forall \sigma{\in}\Sigma.\ \forall b{\in}C.\ \textit{genesis} = \textit{n-cestor}\ (b,\ \textit{Suc}\ n) \wedge$
        $(\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in \textit{best-children}\ (\textit{prev}\ b',\ \sigma)) \longrightarrow$
        $(\forall b''{\in}\textit{GHOST}\ (\{\textit{genesis}\},\ \sigma).\ b \downharpoonleft b'')$
        apply (*rule*, *rule*, *rule*, *rule*)
      proof −
        fix $\sigma\ b\ b''$
        assume $\sigma \in \Sigma$
        and $b \in C$
        and $\textit{genesis} = \textit{n-cestor}\ (b,\ \textit{Suc}\ n) \wedge (\forall b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in \textit{best-children}$
$(\textit{prev}\ b',\ \sigma))$
        and $b'' \in \textit{GHOST}\ (\{\textit{genesis}\},\ \sigma)$
        then have $\textit{genesis} = \textit{n-cestor}\ (\textit{prev}\ b,\ n) \wedge (\forall\ b' \in C.\ b' \downharpoonleft \textit{prev}\ b \longrightarrow b'$
$\in \textit{best-children}\ (\textit{prev}\ b',\ \sigma))$
          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 $\textit{prev}\ b \downharpoonleft b''$
          using ‹$\forall \sigma{\in}\Sigma.\ \forall\ b \in C.\ \textit{genesis} = \textit{n-cestor}\ (b,\ n) \wedge$
            $(\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in \textit{best-children}\ (\textit{prev}\ b',\ \sigma)) \longrightarrow$
            $(\forall b''{\in}\textit{GHOST}\ (\{\textit{genesis}\},\ \sigma).\ b \downharpoonleft b'')$›
          using ‹$\sigma \in \Sigma$› ‹$b \in C$› *prev-type* ‹$b'' \in \textit{GHOST}\ (\{\textit{genesis}\},\ \sigma)$› **by** *auto*
        have $b \in \textit{best-children}\ (\textit{prev}\ b,\ \sigma)$
          using ‹$\textit{genesis} = \textit{n-cestor}\ (b,\ \textit{Suc}\ n) \wedge (\forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in$
$\textit{best-children}\ (\textit{prev}\ b',\ \sigma))$›
          using ‹$b \in C$› *irreflexivity-of-blockchain-membership* **by** *blast*
        then show $b \downharpoonleft b''$
          using ‹$\textit{prev}\ b \downharpoonleft b''$› ‹$b'' \in \textit{GHOST}\ (\{\textit{genesis}\},\ \sigma)$›

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* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in \textit{est}\ `\sigma.\ \forall\ b' \in C.\ b' \downharpoonleft b \longrightarrow b' \in \textit{est}\ `\sigma$
sorry

lemma (**in** *TFG*) *block-membership-is-max-driven* :
$\forall\ \sigma \in \Sigma.\ \forall\ b \in \textit{est}\ `\sigma.$ *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 $\forall\ b' \in C.\ b' \downharpoonright$ *est m* $\longrightarrow$ *weight-measure* (*agreeing-validators* (*block-membership* b', σ')) > *weight-measure* (*disagreeing-validators* (*block-membership* (*est m*), σ'))
using ‹$\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b,\ b'\} \subseteq C \land b' \downharpoonright b$
$\longrightarrow$ *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 ‹$\forall\ \sigma \in \Sigma.\ \forall\ b\ b'.\ \{b,\ b'\} \subseteq C \land b' \downharpoonright b$
$\longrightarrow$ *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