Minimal CBC Casper Isabelle/HOL proofs

LayerX

March 18, 2019

Contents

theory *Strict-Order*

imports *Main*

begin

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

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

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

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

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

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

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

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

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

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

lemma *strict-partial-order-on-finite-non-empty-set-has-maximal* :
strict-partial-order r $\longrightarrow$ *finite* A $\longrightarrow$ $A \neq \emptyset$ $\longrightarrow$ ($\exists\ x.$ *maximal-on* A r x)
proof −
have $\bigwedge n.$ *strict-partial-order* r $\Longrightarrow$ ($\forall\ A.$ *Suc* n = *card* A $\longrightarrow$ *finite* A $\longrightarrow$ $A \neq \emptyset$ $\longrightarrow$ ($\exists\ x.$ *maximal-on* A r x))
proof −
assume *strict-partial-order* r
then have ($\forall a.\ (a,\ a) \notin r$)
by (*simp add*: *strict-partial-order-def irrefl-def*)
fix n
show $\forall\ A.$ *Suc* n = *card* A $\longrightarrow$ *finite* A $\longrightarrow$ $A \neq \emptyset$ $\longrightarrow$ ($\exists\ x.$ *maximal-on* A r x)
apply (*induction* n)
unfolding *maximal-on-def*
using ‹($\forall a.\ (a,\ a) \notin r$)›
apply (*metis card-eq-SucD empty-iff insert-iff*)
proof −
fix n
assume $\forall A.$ *Suc* n = *card* A $\longrightarrow$ *finite* A $\longrightarrow$ $A \neq \emptyset$ $\longrightarrow$ ($\exists x.\ x \in A \wedge (\forall y.\ (x,\ y) \in r \longrightarrow y \notin A)$)
have $\forall B.$ *Suc* (*Suc* n) = *card* B $\longrightarrow$ *finite* B $\longrightarrow$ $B \neq \emptyset$ $\longrightarrow$ ($\exists\ A'\ b.\ B = A' \cup \{b\} \wedge$ *card* A' = *Suc* n $\wedge\ b \notin A'$)
by (*metis Un-commute add-diff-cancel-left′ card-gt-0-iff card-insert-disjoint card-le-Suc-iff insert-is-Un not-le not-less-eq-eq plus-1-eq-Suc*)
then have $\forall B.$ *Suc* (*Suc* n) = *card* B $\longrightarrow$ *finite* B $\longrightarrow$ $B \neq \emptyset$ $\longrightarrow$ ($\exists\ A'\ b.\ B = A' \cup \{b\} \wedge$ *card* A' = *Suc* n $\wedge$ *finite* $A' \wedge A' \neq \emptyset \wedge b \notin A'$)
by (*metis card-gt-0-iff zero-less-Suc*)
then have $\forall B.$ *Suc* (*Suc* n) = *card* B $\longrightarrow$ *finite* B $\longrightarrow$ $B \neq \emptyset$
$\longrightarrow$ ($\exists\ A'\ b\ x.\ B = A' \cup \{b\} \wedge b \notin A' \wedge x \in A' \wedge (\forall y.\ (x,\ y) \in r \longrightarrow y \notin A')$)
using ‹$\forall A.$ *Suc* n = *card* A $\longrightarrow$ *finite* A $\longrightarrow$ $A \neq \emptyset$ $\longrightarrow$ ($\exists x.\ x \in A \wedge (\forall y.\ (x,\ y) \in r \longrightarrow y \notin A)$)›
by *metis*

then show $\forall B.\ Suc\ (Suc\ n) = card\ B \longrightarrow finite\ B \longrightarrow B \neq \emptyset \longrightarrow (\exists x.\ x \in B \wedge (\forall y.\ (x, y) \in r \longrightarrow y \notin B))$
by (*metis* (*no-types, lifting*) *Un-insert-right* ‹$\forall a.\ (a, a) \notin r$› ‹*strict-partial-order r*› *insertE insert-iff strict-partial-order-def sup-bot.right-neutral transE*)
qed
qed
then show *?thesis*
by (*metis card.insert-remove finite.cases*)
qed

lemma *strict-partial-order-has-at-most-one-maximum* :
strict-partial-order r
$\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset$
$\longrightarrow$ *is-singleton* $\{x.\ maximum\text{-}on\ A\ r\ x\}$
proof (*rule ccontr*)
assume $\neg$ (*strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow$ *is-singleton* $\{x.\ maximum\text{-}on\ A\ r\ x\}$)
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow \neg$ *is-singleton* $\{x.\ maximum\text{-}on\ A\ r\ x\}$
by *simp*
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow (\exists\ x1\ x2.\ x1 \neq x2 \wedge \{x1, x2\} \subseteq \{x.\ maximum\text{-}on\ A\ r\ x\})$
by (*meson empty-subsetI insert-subset is-singletonI'*)
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow (\exists\ x1\ x2.\ x1 \neq x2 \wedge \{x1, x2\} \subseteq \{x \in A.\ \forall\ y.\ y \in A \longrightarrow (y, x) \in r \vee x = y\})$
by (*simp add*: *maximum-on-def upper-bound-on-def*)
then have *strict-partial-order r* $\longrightarrow \{x.\ maximum\text{-}on\ A\ r\ x\} \neq \emptyset \longrightarrow (\exists\ x1\ x2.\ x1 \neq x2 \wedge \{x1, x2\} \subseteq A \wedge (\forall\ y.\ y \in A \longrightarrow (y, x1) \in r \vee x1 = y) \wedge (\forall\ y.\ y \in A \longrightarrow (y, x2) \in r \vee x2 = y))$
by *auto*
then show *False*
using *strict-partial-order-def*

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

lemma *strict-linear-order-on-finite-non-empty-set-has-one-maximum* :
strict-linear-order-on A r $\longrightarrow$ *finite A* $\longrightarrow A \neq \emptyset \longrightarrow$ *is-singleton* $\{x.\ maximum\text{-}on\ A\ r\ x\}$
using *strict-linear-order-is-strict-partial-order strict-partial-order-on-finite-non-empty-set-has-maximal*

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

end

1 CBC Casper

theory *CBCCasper*

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

begin

notation *Set.empty* (∅)

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
  Σ*i* :: (*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* $\Rightarrow$ *real*
and t :: *real*
fixes C :: *consensus-value set*
and ε :: *message set* $\Rightarrow$ *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* $\Rightarrow$ *consensus-value set*) $\Rightarrow$ *bool*
where
is-valid-estimator $e = (\forall \sigma \in \Sigma.\ e\ \sigma \in Pow\ C - \{\emptyset\})$

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

lemma Σ*i-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-*Σ*i*: $Mi\ (V,C,\varepsilon)\ n \subseteq Mi\ (V,C,\varepsilon)\ (n+1) \Longrightarrow \Sigma i\ (V,C,\varepsilon)\ (n+1) \subseteq \Sigma i\ (V,C,\varepsilon)\ (n+2)$
by *auto*

lemma Σ*i-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-*Σ*i Suc-eq-plus1* Σ*i-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 Σ*i-monotonic* Σ*i-subset-to-Mi* **apply** *blast*
apply *auto*
done

lemma Σ*i-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 Σ*i-monotonic*

by (*metis Suc-eq-plus1 lift-Suc-mono-le*)

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

lemma *message-is-in-Mi* :
$\forall\ m \in M.\ \exists\ n \in \mathbb{N}.\ m \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
apply (*simp add*: *M-def* Σi*.elims*)
by (*metis Nats-1 Nats-add One-nat-def diff-Suc-1 plus-1-eq-Suc*)

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

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

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

lemma *message-in-state-is-valid* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \wedge m \in \sigma \longrightarrow\ m \in M$
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.Σi-monotonic PowD Suc-diff-Suc* ‹$\sigma \in \Sigma \wedge m \in \sigma$› *add-leE diff-add diff-le-self gr0I mem-Collect-eq plus-1-eq-Suc state-is-in-pow-Mi subsetCE zero-less-diff*)
qed

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

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

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

lemma 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*: $\bigwedge w.\ w \in$ *range* $W \Longrightarrow w > 0$
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* $\Sigma i.simps(1)$ **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*) Σi*-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* $\langle\sigma \in \Sigma\rangle$ **by** *auto*
have *finite* σ
using *state-is-finite* $\langle\sigma \in \Sigma\rangle$ **by** *auto*
moreover have $\exists\ m.$ *sender* $m \in V \wedge$ *est* $m \in \varepsilon\ \sigma \wedge$ *justification* $m = \sigma$
using *est.simps sender.simps justification.simps V-type* $\langle\varepsilon\ \sigma \neq \emptyset\rangle$ $\langle$*finite* $\sigma\rangle$
by (*metis all-not-in-conv finite-list*)
moreover have $\varepsilon\ \sigma \subseteq C$
using *estimates-are-subset-of-C* Σ*i-is-subset-of-*Σ $\langle n \in \mathbb{N}\rangle$ $\langle\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n\rangle$ **by** *blast*
ultimately show $\exists\ m.$ *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 $\langle\sigma \in \Sigma i\ (V,\ C,\ \varepsilon)\ n\rangle$ **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* $\langle m' \in Mi\ (V,\ C,\ \varepsilon)\ (Suc\ 0) \wedge$ *justification* $m' = \{m\}\rangle$ **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* ⇒ *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*) ⇒ *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* ⇒ *message* ⇒ *bool*
  where
    justified m1 m2 = (*m1* ∈ *justification m2*)

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

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

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

lemma (**in** *Protocol*) *equivocating-validators-type* :
$\forall\ \sigma \in \Sigma.$ *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 $\sigma = \{v \in V.$ *is-equivocating* $\sigma\ v\}$

lemma (**in** *Protocol*) *equivocating-validators-is-equivalent-to-paper* :
$\forall\ \sigma \in \Sigma.$ *equivocating-validators* $\sigma =$ *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* :
$\forall\ \sigma\ \sigma'\ v.\ \sigma \in \Sigma \wedge \sigma' \in \Sigma \wedge$ *is-future-state* $(\sigma, \sigma') \wedge v \in V$
$\longrightarrow v \in$ *equivocating-validators* σ
$\longrightarrow v \in$ *equivocating-validators* σ'
apply (*simp add*: *equivocating-validators-def is-equivocating-def*)
using *observed-def* **by** *fastforce*

definition (**in** *Params*) *weight-measure* :: *validator set* $\Rightarrow$ *real*
where

weight-measure v-set = *Sum* (*W* ‘*v-set*)

lemma (**in** *Protocol*) *weight-measure-comparison-strict-subset-gte* :
finite $A \Longrightarrow$ *finite* $B \Longrightarrow B \subseteq A \Longrightarrow$ *weight-measure* $A \geq$ *weight-measure* B
apply (*simp add*: *weight-measure-def*)
using *W-type*
by (*smt Diff-iff finite-imageI subsetCE subset-UNIV subset-image-iff sum-mono2*)

lemma (**in** *Protocol*) *weight-measure-comparison-stritct-subset-gt* :
finite $A \Longrightarrow$ *finite* $B \Longrightarrow B \subset A \Longrightarrow$ *weight-measure* $A >$ *weight-measure* B
apply (*simp add*: *weight-measure-def*)
using *W-type*
oops

lemma (**in** *Protocol*) *weight-measure-gt-set-difference* :
finite $A \Longrightarrow$ *finite* $B \Longrightarrow B \neq \emptyset \Longrightarrow$ *weight-measure* $A >$ *weight-measure* $(A - B)$

oops

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

$equivocation\text{-}fault\text{-}weight\ \sigma = weight\text{-}measure\ (equivocating\text{-}validators\ \sigma)$

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

definition (**in** *Params*) *is-faults-lt-threshold* :: *state* ⇒ *bool*
where
$is\text{-}faults\text{-}lt\text{-}threshold\ \sigma = (equivocation\text{-}fault\text{-}weight\ \sigma < t)$

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

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 = $\{(m1,\ m2).\ \{m1,\ m2\} \subseteq M \wedge$ *justified m1 m2*$\}$

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

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

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

lemma (**in** *Protocol*) *justification-is-strict-partial-order-on-M* :
strict-partial-order message-justification

apply (*simp add*: *strict-partial-order-def*)
by (*simp add*: *irreflexivity-of-justifications transitivity-of-justifications*)

lemma (**in** *Protocol*) *monotonicity-of-justifications* :
$\forall$ *m m′ σ*. $m \in M \wedge \sigma \in \Sigma \wedge$ *justified m′ m* $\longrightarrow$ *justification m′* $\subseteq$ *justification m*
apply *simp*
by (*meson M-type justified-def message-in-state-is-valid state-is-in-pow-Mi*)

lemma (**in** *Protocol*) *strict-monotonicity-of-justifications* :
$\forall$ *m m′ σ*. $m \in M \wedge \sigma \in \Sigma \wedge$ *justified m′ m* $\longrightarrow$ *justification m′* $\subset$ *justification m*
by (*metis M-type message-cannot-justify-itself justified-def message-in-state-is-valid monotonicity-of-justifications psubsetI*)

lemma (**in** *Protocol*) *justification-implies-different-messages* :
$\forall$ *m m′*. $m \in M \wedge m' \in M \longrightarrow$ *justified m′ m* $\longrightarrow m \neq m'$
using *message-cannot-justify-itself* **by** *auto*

lemma (**in** *Protocol*) *only-valid-message-is-justified* :
$\forall\ m \in M.\ \forall\ m'.$ *justified m′ m* $\longrightarrow m' \in M$
apply (*simp add*: *justified-def*)
using *Params.M-type message-in-state-is-valid* **by** *blast*

lemma (**in** *Protocol*) *justified-message-exists-in-Mi-n-minus-1* :
$\forall$ *n m m′*. $n \in \mathbb{N}$
$\longrightarrow$ *justified m′ m*
$\longrightarrow m \in Mi\ (V,\ C,\ \varepsilon)\ n$
$\longrightarrow m' \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
proof −
have $\forall$ *n m m′*. *justified m′ m*
$\longrightarrow m \in Mi\ (V,\ C,\ \varepsilon)\ n$
$\longrightarrow m \in M \wedge m' \in M$
$\longrightarrow m' \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix *n m m′*
assume *justified m′ m*
assume $m \in Mi\ (V,\ C,\ \varepsilon)\ n$
assume $m \in M \wedge m' \in M$
then have *justification m* $\in \Sigma i\ (V,C,\varepsilon)\ n$
using *Mi.simps* ‹$m \in Mi\ (V,\ C,\ \varepsilon)\ n$› **by** *blast*
then have *justification m* $\in$ *Pow* $(Mi\ (V,C,\varepsilon)\ (n - 1))$
by (*metis* (*no-types*, *lifting*) *Suc-diff-Suc* Σi*.simps*(*1*) Σi*-subset-Mi* ‹*justified m′ m*› *add-leE diff-add diff-le-self empty-iff justified-def neq0-conv plus-1-eq-Suc singletonD subsetCE*)
show $m' \in Mi\ (V,\ C,\ \varepsilon)\ (n - 1)$
using ‹*justification m* $\in$ *Pow* $(Mi\ (V,\ C,\ \varepsilon)\ (n - 1))$› ‹*justified m′ m*› *justified-def* **by** *auto*

qed
then show *?thesis*
by (*metis* (*no-types*, *lifting*) *M-def UN-I only-valid-message-is-justified*)
qed

lemma (**in** *Protocol*) *monotonicity-of-card-of-justification* :
$\forall\ m\ m'.\ m \in M$
$\longrightarrow$ *justified* $m'\ m$
$\longrightarrow$ *card* (*justification* m') $<$ *card* (*justification* m)
by (*meson M-type Protocol.strict-monotonicity-of-justifications Protocol-axioms justification-is-finite psubset-card-mono*)

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

lemma (**in** *Protocol*) *subset-of-M-have-minimal-of-justification* :
$\forall\ S \subseteq M.\ S \neq \emptyset \longrightarrow (\exists\ m\text{-}min \in S.\ \forall\ m.$ *justified* $m\ m\text{-}min \longrightarrow m \notin S)$
by (*metis justification-is-well-founded-on-M wfp-on-imp-has-min-elt wfp-on-mono*)

lemma (**in** *Protocol*) *message-in-state-is-strict-subset-of-the-state* :
$\forall\ \sigma \in \Sigma.\ \forall\ m \in \sigma.$ *justification* $m \subset \sigma$

using *justification-implies-different-messages justified-def message-in-state-is-valid state-is-in-pow-Mi* **by** *fastforce*

end

3 Latest Message

theory *LatestMessage*

imports *Main CBCCasper MessageJustification Libraries/LaTeXsugar*

begin

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

irreflexivity-of-justifications transitivity-of-justifications)

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

by *blast*

lemma (**in** *Protocol*) *latest-message-is-maximal-element-of-justification* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *latest-messages* $\sigma\ v = \{m.$ *maximal-on* (*from-sender* (v, σ)) *message-justification* $m\}$
apply (*simp add*: *latest-messages-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* (*latest-messages* $\sigma\ v$))
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* (*latest-messages* $\sigma\ v$)
using *latest-message-is-maximal-element-of-justification*
observed-non-equivocating-validators-def observed-non-equivocating-validators-type

by *fastforce*
qed

definition *latest-estimates* :: *state* $\Rightarrow$ *validator* $\Rightarrow$ *consensus-value set*
where
latest-estimates σ v = $\{$*est m* | *m*. *m* $\in$ *latest-messages* σ $v\}$

lemma (**in** *Protocol*) *latest-estimates-type* :
$\forall$ σ v. $\sigma \in \Sigma \land v \in V \longrightarrow$ *latest-estimates* σ $v \subseteq C$
using *M-type Protocol.latest-messages-type-for-state Protocol-axioms latest-estimates-def*
by *fastforce*

lemma (**in** *Protocol*) *latest-estimates-from-non-observed-validator-is-empty* :
$\forall$ σ v. $\sigma \in \Sigma \land v \in V \land v \notin$ *observed* $\sigma \longrightarrow$ *latest-estimates* σ $v = \emptyset$
using *latest-estimates-def latest-messages-from-non-observed-validator-is-empty*
by *auto*

definition *latest-messages-from-non-equivocating-validators* :: *state* $\Rightarrow$ *validator* $\Rightarrow$ *message set*
where
latest-messages-from-non-equivocating-validators σ v = (*if* $v \in$ *equivocating-validators* σ *then* $\emptyset$ *else latest-messages* σ v)

lemma (**in** *Protocol*) *latest-messages-from-non-equivocating-validators-type* :
$\forall$ σ v. $\sigma \in \Sigma \land v \in V \longrightarrow$ *latest-messages-from-non-equivocating-validators* σ v $\subseteq M$
by (*simp add*: *latest-messages-type-for-state latest-messages-from-non-equivocating-validators-def*)

lemma (**in** *Protocol*) *observed-non-equivocating-validator-has-one-latest-message* :
$\forall$ $\sigma \in \Sigma$. $\forall$ $v \in$ *observed-non-equivocating-validators* σ.
is-singleton (*latest-messages-from-non-equivocating-validators* σ v)
using *observed-non-equivocating-validators-have-one-latest-message*
by (*simp add*: *latest-messages-from-non-equivocating-validators-def observed-non-equivocating-validators-def*)

lemma (**in** *Protocol*) *sender-of-latest-message-of-observed-non-equivocating-validator*:

$\forall$ $\sigma \in \Sigma$. $\forall$ $v \in$ *observed-non-equivocating-validators* σ. *sender* (*the-elem* (*latest-messages-from-non-equivocati*

$\sigma\ v)) = v$
using *observed-non-equivocating-validator-has-one-latest-message latest-messages-from-non-equivocating-validators-def latest-messages-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*) *latest-message-of-observed-non-equivocating-validator-is-in-the-state*:

$\forall\ \sigma \in \Sigma.\ \forall\ v \in$ *observed-non-equivocating-validators* σ. *the-elem* (*latest-messages-from-non-equivocating-valid*
$\sigma\ v) \in \sigma$
using *observed-non-equivocating-validator-has-one-latest-message latest-messages-from-non-equivocating-validators-def latest-messages-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 *latest-estimates-from-non-equivocating-validators* :: *state* $\Rightarrow$ *validator* $\Rightarrow$ *consensus-value set*
where

latest-estimates-from-non-equivocating-validators $\sigma\ v$ = *est* '*latest-messages-from-non-equivocating-validator*
$\sigma\ v$

lemma (**in** *Protocol*) *latest-estimates-from-non-equivocating-validators-type* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \land v \in V \longrightarrow$ *latest-estimates-from-non-equivocating-validators* $\sigma\ v$
$\in$ *Pow C*
using *Protocol.latest-estimates-type Protocol-axioms latest-estimates-def latest-estimates-from-non-equivocati latest-messages-from-non-equivocating-validators-def*
using *M-type latest-messages-from-non-equivocating-validators-type* **by** *fastforce*

lemma (**in** *Protocol*) *latest-estimates-from-non-equivocating-validators-from-non-observed-validator-is-empty* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \land v \in V \land v \notin$ *observed* $\sigma \longrightarrow$ *latest-estimates-from-non-equivocating-validators*
$\sigma\ v = \emptyset$
by (*simp add*: *latest-estimates-from-non-equivocating-validators-def latest-messages-from-non-equivocating-va latest-messages-from-non-observed-validator-is-empty*)

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

lemma (**in** *Protocol*) *observed-non-equivocating-validator-has-one-latest-estimate* :

$\forall\ \sigma \in \Sigma.\ \forall\ v \in$ *observed-non-equivocating-validators* σ.
is-singleton (*latest-estimates-from-non-equivocating-validators* $\sigma\ v$)

using *observed-non-equivocating-validator-has-one-latest-message*
apply (*simp add*: *latest-estimates-from-non-equivocating-validators-def*)
using *image-of-singleton-is-singleton*
by *blast*

definition *latest-justifications-from-non-equivocating-validators* :: *state* ⇒ *validator* ⇒ *state set*
where
latest-justifications-from-non-equivocating-validators σ v = *justification* '*latest-messages-from-non-equivocati*
σ v

lemma (**in** *Protocol*) *latest-justifications-from-non-equivocating-validators-type* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \land v \in V \longrightarrow$ *latest-justifications-from-non-equivocating-validators* $\sigma\ v \subseteq \Sigma$
using *M-type latest-messages-from-non-equivocating-validators-type latest-justifications-from-non-equivocating-validators-def* **by** *auto*

lemma (**in** *Protocol*) *observed-non-equivocating-validator-has-one-latest-justification* :
$\forall\ \sigma \in \Sigma.\ v \in$ *observed-non-equivocating-validators* σ
$\longrightarrow$ *is-singleton* (*latest-justifications-from-non-equivocating-validators* σ v)
using *observed-non-equivocating-validator-has-one-latest-message*
apply (*simp add*: *latest-justifications-from-non-equivocating-validators-def*)
using *image-of-singleton-is-singleton*
by *blast*

lemma (**in** *Protocol*) *latest-justification-is-strict-subset-of-state* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \land v \in V \longrightarrow (\forall\ \sigma' \in$ *latest-justifications-from-non-equivocating-validators* $\sigma\ v.\ \sigma' \subset \sigma)$
apply (*simp add*: *latest-justifications-from-non-equivocating-validators-def latest-messages-from-non-equivocating-validators-def*)
using *latest-messages-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* $(\sigma,$
$\sigma') \wedge \sigma \neq \sigma'$
$\wedge (\nexists\ \sigma''.\ \sigma'' \in \Sigma \wedge$ *is-future-state* $(\sigma, \sigma'') \wedge$ *is-future-state* $(\sigma'', \sigma') \wedge \sigma \neq$
$\sigma'' \wedge \sigma'' \neq \sigma')\}$

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

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

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

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

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

have *justification* $m \subseteq \sigma$
using *immediately-next-message-def*
by (*metis* (*no-types*, *lifting*) ‹*immediately-next-message* (σ, m)› *case-prod-conv*)
hence *justification* $m \subseteq \sigma \cup \{m\}$
by *blast*
moreover have $\bigwedge m'. \ finite \ \sigma \wedge m' \in \sigma \Longrightarrow justification \ m' \subseteq \sigma$
using ‹$\sigma \in \Sigma i$ (V, C, 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, ε) $(n + 1)$
using ‹$\bigwedge m'. \ finite \ \sigma \wedge m' \in \sigma \Longrightarrow justification \ m' \subseteq \sigma \cup \{m\}$› ‹$\sigma \cup \{m\} \in Pow$ $(Mi$ (V, C, ε) $n)$› ‹*justification* $m \subseteq \sigma \cup \{m\}$›
‹$\sigma \in \Sigma i$ (V, C, ε) n› *si* **by** *auto*
qed

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

$\forall \sigma \in \Sigma i$ (V,C,ε) n. $\forall m \in Mi$ (V,C,ε) n. *immediately-next-message* $(\sigma,m) \longrightarrow \sigma \cup \{m\} \in \Sigma i$ (V,C,ε) $(n+1)$
apply (*cases n*)
apply *auto*[*1*]
using *state-transition-by-immediately-next-message-of-same-depth-non-zero*

by (*metis le-add1 plus-1-eq-Suc*)

lemma (**in** *Params*) *past-state-exists-in-same-depth* :
$\forall\ \sigma\ \sigma'.\ \sigma' \in \Sigma i\ (V,C,\varepsilon)\ n \longrightarrow \sigma \subseteq \sigma' \longrightarrow \sigma \in \Sigma \longrightarrow \sigma \in \Sigma i\ (V,C,\varepsilon)\ n$
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 $\langle\sigma \in \Sigma\rangle$ *state-is-finite state-is-in-pow-Mi* **by** *blast*
moreover have $\sigma \in Pow\ (Mi\ (V,\ C,\ \varepsilon)\ (n - 1))$
using $\langle\sigma \subseteq \sigma'\rangle$
by (*smt Pow-iff Suc-eq-plus1* Σ*i-monotonic* Σ*i-subset-Mi* $\langle\sigma' \in \Sigma i\ (V,\ C,\ \varepsilon)\ n\rangle$ *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 $\sigma\ m$
assume $\sigma \in \Sigma$
and $m \in M$

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

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

lemma (**in** *Protocol*) *state-transition-only-made-by-immediately-next-message*:
$\forall\ \sigma \in\Sigma.\ \forall\ m \in M.\ \sigma \cup \{m\} \in \Sigma \wedge m \notin \sigma \longleftrightarrow$ *immediately-next-message* (σ,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-next-message*($\sigma \cap \sigma'$, m))
by (*metis Int-subset-iff psubsetI strict-subset-of-state-have-immediately-next-messages subsetI*)
then have $\forall$ $\sigma \in \Sigma$. $\forall$ $\sigma' \in \Sigma$. $\neg$ $\sigma \subseteq \sigma' \longrightarrow$ ($\exists$ $m \in \sigma - (\sigma \cap \sigma')$. *immediately-next-message*(σ', m))
apply (*simp add*: *immediately-next-message-def*)
by *blast*
then have $\forall$ $\sigma \in \Sigma$. $\forall$ $\sigma' \in \Sigma$. $\neg$ $\sigma \subseteq \sigma' \longrightarrow$ ($\exists$ $m \in \sigma - \sigma'$. $\sigma' \cup \{m\} \in \Sigma$)
using *state-transition-by-immediately-next-message*
by (*metis DiffD1 DiffD2 DiffI IntI message-in-state-is-valid*)
have $\forall$ $\sigma \in \Sigma$. $\forall$ $\sigma' \in \Sigma$. $\neg$ $\sigma \subseteq \sigma' \longrightarrow$ $\sigma \cup \sigma' \in \Sigma$
proof $-$
have $\forall$ $\sigma \in \Sigma$. $\forall$ $\sigma' \in \Sigma$. $\neg$ $\sigma \subseteq \sigma' \longrightarrow$ *card* $(\sigma - \sigma') > 0$
by (*meson Diff-eq-empty-iff card-0-eq finite-Diff gr0I state-is-finite*)

have $\forall\ n.\ \forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* $n =$ *card* $(\sigma - \sigma') \longrightarrow\ \sigma \cup \sigma' \in \Sigma$
apply (*rule*)
proof −
fix n
show $\forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* $n =$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
apply (*induction n*)
apply (*rule*, *rule*, *rule*)
proof −
fix $\sigma\ \sigma'$
assume $\sigma \in \Sigma$ **and** $\sigma' \in \Sigma$ **and** $\neg\ \sigma \subseteq \sigma' \wedge$ *Suc 0* $=$ *card* $(\sigma - \sigma')$
then have *is-singleton* $(\sigma - \sigma')$
by (*simp add*: *is-singleton-altdef*)
then have $\{$*the-elem* $(\sigma - \sigma')\} \cup \sigma' \in \Sigma$
using ‹$\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)$› ‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$›
by (*metis Un-commute* ‹$\neg\ \sigma \subseteq \sigma' \wedge$ *Suc 0* $=$ *card* $(\sigma - \sigma')$› *is-singleton-the-elem singletonD*)
then show $\sigma \cup \sigma' \in \Sigma$
by (*metis Un-Diff-cancel2* ‹*is-singleton* $(\sigma - \sigma')$› *is-singleton-the-elem*)

next
show $\bigwedge n.\ \forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* $n =$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma \Longrightarrow \forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* (*Suc* n) $=$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
apply (*rule*, *rule*, *rule*)
proof −
fix $n\ \sigma\ \sigma'$
assume $\forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* $n =$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$ **and** $\sigma \in \Sigma$ **and** $\sigma' \in \Sigma$ **and** $\neg\ \sigma \subseteq \sigma' \wedge$ *Suc* (*Suc* n) $=$ *card* $(\sigma - \sigma')$
have $\forall\ m \in \sigma - \sigma'.\ \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge$ *Suc* $n =$ *card* $(\sigma - (\sigma' \cup \{m\}))$
using ‹$\neg\ \sigma \subseteq \sigma' \wedge$ *Suc* (*Suc* n) $=$ *card* $(\sigma - \sigma')$›
by (*metis Diff-eq-empty-iff Diff-insert Un-insert-right* ‹$\sigma \in \Sigma$› *add-diff-cancel-left′ card-0-eq card-Suc-Diff1 finite-Diff nat.simps*(*3*) *plus-1-eq-Suc state-is-finite sup-bot.right-neutral*)
have $\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma$
using ‹$\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)$› ‹$\sigma \in \Sigma$› ‹$\sigma' \in \Sigma$› ‹$\neg\ \sigma \subseteq \sigma' \wedge$ *Suc* (*Suc* n) $=$ *card* $(\sigma - \sigma')$›
by *blast*
then have $\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma \wedge \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge$ *Suc* $n =$ *card* $(\sigma - (\sigma' \cup \{m\}))$
using ‹$\forall\ m \in \sigma - \sigma'.\ \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge$ *Suc* $n =$ *card* $(\sigma - (\sigma' \cup \{m\}))$›
by *simp*
then show $\sigma \cup \sigma' \in \Sigma$
using ‹$\forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* $n =$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$›
by (*smt Un-Diff-cancel Un-commute Un-insert-right* ‹$\sigma \in \Sigma$› *insert-absorb2 mk-disjoint-insert sup-bot.right-neutral*)
qed

qed
qed
then show *?thesis*
by (*meson* ‹$\forall \sigma \in \Sigma.\ \forall \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\, m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)$› *card-Suc-Diff1 finite-Diff state-is-finite*)
qed
then show *?thesis*
using *False* ‹$\sigma 1 \in \Sigma$› ‹$\sigma 2 \in \Sigma$› **by** *blast*
qed
qed

lemma (**in** *Protocol*) *union-of-finite-set-of-states-is-state* :
$\forall\ \sigma\text{-}set \subseteq \Sigma.\ finite\ \sigma\text{-}set \longrightarrow \bigcup\ \sigma\text{-}set \in \Sigma$
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* ‹$\sigma\text{-}set \subseteq \Sigma$› **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*:
$\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ is\text{-}future\text{-}state\ (\sigma, \sigma') \wedge \sigma \neq \sigma' \longrightarrow (\exists\ m \in \sigma' - \sigma.\ immediately\text{-}next\text{-}message$

$(\sigma, m))$
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* (σ, σ')› *insert-subset*
message-in-state-is-valid)
qed
have $\forall\ \sigma\ \sigma'.\ (\sigma, \sigma') \in$ *minimal-transitions* $\wedge\ \neg$ *is-singleton* $(\sigma' - \sigma) \longrightarrow$
($\exists\ \sigma''.\ \sigma'' \in \Sigma\ \wedge$ *is-future-state* $(\sigma, \sigma'')\ \wedge$ *is-future-state* $(\sigma'', \sigma')\ \wedge\ \sigma$
$\neq \sigma'' \wedge \sigma'' \neq \sigma'$)
apply (*rule*, *rule*, *rule*)

proof −
fix σ σ'
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 ‹$\forall$ σ σ'. $(\sigma, \sigma') \in$ *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* :
$\forall$ σ σ'. $(\sigma, \sigma') \in$ *minimal-transitions* $\longrightarrow \sigma \cup \{$*the-elem* $(\sigma' - \sigma)\} = \sigma'$
apply (*rule*, *rule*, *rule*)
proof −
fix σ σ'
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*) *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* $\Rightarrow$ *state set*
where
futures $\sigma = \{\sigma' \in \Sigma t.$ *is-future-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* σ
apply (*simp add*: *futures-def*) **by** *auto*

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

theorem (**in** *Protocol*) *n-party-common-futures* :
$\forall\ \sigma$-*set*. σ-*set* $\subseteq \Sigma t$
$\longrightarrow$ *finite* σ-*set*

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

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

lemma (**in** *Protocol*) *backword-consistency* :
$\forall\ \sigma'\ \sigma.\ \sigma' \in \Sigma t \wedge \sigma \in \Sigma t$
$\longrightarrow \sigma' \in futures\ \sigma$
$\longrightarrow$ *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* ⇒ *bool*
where
state-properties-are-inconsistent p-set = ($\forall$ $\sigma \in \Sigma$. $\neg$ ($\forall$ $p \in$ *p-set*. p σ))

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

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

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

definition (**in** *Protocol*) *naturally-corresponding-state-property* :: *consensus-value-property* ⇒ *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* ⇒ *bool*
where
consensus-value-properties-are-consistent q-set = $(\exists\ c \in C.\ \forall\ q \in q\text{-}set.\ q\ c)$

lemma (**in** *Protocol*) *naturally-corresponding-consistency* :
$\forall$ *q-set*. *state-properties-are-consistent* {*naturally-corresponding-state-property q* | *q. q* ∈ *q-set*}
⟶ *consensus-value-properties-are-consistent q-set*
apply (*rule*, *rule*)
proof −
fix *q-set*
have
state-properties-are-consistent {*naturally-corresponding-state-property q* | *q. q* ∈ *q-set*}
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ p \in \{\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q\ c \mid q.\ q \in q\text{-}set\}.\ p\ \sigma)$
by (*simp add*: *naturally-corresponding-state-property-def state-properties-are-consistent-def*)
moreover have
$(\exists\ \sigma \in \Sigma.\ \forall\ p \in \{\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q\ c \mid q.\ q \in q\text{-}set\}.\ p\ \sigma)$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ q' \in q\text{-}set.\ (\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q'\ c)\ \sigma)$

by (*metis* (*mono-tags*, *lifting*) *mem-Collect-eq*)
moreover have
$(\exists\ \sigma \in \Sigma.\ \forall\ q \in q\text{-}set.\ (\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q\ c)\ \sigma)$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ q' \in q\text{-}set.\ \forall\ c \in \varepsilon\ \sigma.\ q'\ c)$
by *blast*
moreover have
$(\exists\ \sigma \in \Sigma.\ \forall\ q \in q\text{-}set.\ \forall\ c \in \varepsilon\ \sigma.\ q\ c)$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ c \in \varepsilon\ \sigma.\ \forall\ q' \in q\text{-}set.\ q'\ c)$
by *blast*
moreover have
$(\exists\ \sigma \in \Sigma.\ \forall\ c \in \varepsilon\ \sigma.\ \forall\ q \in q\text{-}set.\ q\ c)$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \exists\ c \in \varepsilon\ \sigma.\ \forall\ q' \in q\text{-}set.\ q'\ c)$
by (*meson all-not-in-conv estimates-are-non-empty*)
moreover have
$(\exists\ \sigma \in \Sigma.\ \exists\ c \in \varepsilon\ \sigma.\ \forall\ q \in q\text{-}set.\ q\ c)$
$\longrightarrow (\exists\ c \in C.\ \forall\ q' \in q\text{-}set.\ q'\ c)$
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* :
$\forall\ \sigma\text{-}set.\ \sigma\text{-}set \subseteq \Sigma t$
$\longrightarrow$ *finite* σ*-set*
$\longrightarrow$ *is-faults-lt-threshold* $(\bigcup\ \sigma\text{-}set)$
$\longrightarrow$ *consensus-value-properties-are-consistent* $(\bigcup\ \{$*consensus-value-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\})$
apply (*rule*, *rule*, *rule*, *rule*)
proof −
fix σ*-set*

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

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

lemma (**in** *Protocol*) *negation-is-not-decided-by-other-validator* :
∀ *σ-set*. *σ-set* ⊆ Σ*t*
⟶ *finite σ-set*
⟶ *is-faults-lt-threshold* ($\bigcup$ *σ-set*)
⟶ (∀ *σ σ′ p*. {*σ*, *σ′*} ⊆ *σ-set* ∧ *p* ∈ *consensus-value-property-decisions σ*
⟶ *consensus-value-property-not p* ∉ *consensus-value-property-decisions σ′*)
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)

proof −
fix *σ-set σ σ′ p*
assume $\sigma\text{-}set \subseteq \Sigma t$ **and** *finite σ-set* **and** *is-faults-lt-threshold* ($\bigcup\sigma\text{-}set$) **and** $\{\sigma, \sigma'\} \subseteq \sigma\text{-}set \wedge p \in$ *consensus-value-property-decisions σ*
hence $\exists\ \sigma.\ \sigma \in \Sigma t \wedge \sigma \in \bigcap\ \{futures\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\}$
using *n-party-common-futures-exists* **by** *meson*
then obtain σ'' **where** $\sigma'' \in \Sigma t \wedge \sigma'' \in \bigcap\ \{futures\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\}$ **by** *auto*
hence *state-property-is-decided* (*naturally-corresponding-state-property p*, σ'')
using ‹$\{\sigma, \sigma'\} \subseteq \sigma\text{-}set \wedge p \in$ *consensus-value-property-decisions σ*› *consensus-value-property-decisions-def consensus-value-property-is-decided-def*
using ‹$\sigma\text{-}set \subseteq \Sigma t$› *forward-consistency* **by** *fastforce*
have $\sigma'' \in$ *futures σ′*
using ‹$\sigma'' \in \Sigma t \wedge \sigma'' \in \bigcap\ \{futures\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\}$› ‹$\{\sigma, \sigma'\} \subseteq \sigma\text{-}set \wedge p \in$ *consensus-value-property-decisions σ*›
by *auto*
hence ¬ *state-property-is-decided* (*state-property-not* (*naturally-corresponding-state-property p*), σ′)

using *backword-consistency* ‹*state-property-is-decided* (*naturally-corresponding-state-property p*, σ'')›
using ‹$\sigma'' \in \Sigma t \wedge \sigma'' \in \bigcap$-*Collect* (*futures σ*) ($\sigma \in \sigma\text{-}set$)› ‹$\sigma\text{-}set \subseteq \Sigma t$› ‹$\{\sigma, \sigma'\} \subseteq \sigma\text{-}set \wedge p \in$ *consensus-value-property-decisions σ*› **by** *auto*
then show *consensus-value-property-not p* $\notin$ *consensus-value-property-decisions σ′*
apply (*simp add*: *consensus-value-property-decisions-def consensus-value-property-is-decided-def naturally-corresponding-state-property-def state-property-is-decided-def*)
using *Σt-def estimates-are-non-empty futures-def* **by** *fastforce*
qed

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

have *state-property-is-decided* (*naturally-corresponding-state-property* ($\lambda c.$ ($\neg$ *p* *c*)), σ'')
using ‹($\lambda c.$ ($\neg$ *p* *c*)) $\in \bigcup$ {*consensus-value-property-decisions* σ' | σ'. σ' $\in$ σ*-set*}› *consensus-value-property-decisions-def consensus-value-property-is-decided-def*

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

lemma (**in** *Protocol*) *two-party-consensus-safety-for-consensus-value-property* :
$\forall$ $\sigma 1$ $\sigma 2$. $\sigma 1 \in \Sigma t \wedge \sigma 2 \in \Sigma t$
$\longrightarrow$ *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*. σ $\in$ σ*-set* $\wedge$ *p-set* $\in$ *Pow* ($\bigcup$ {*consensus-value-property-decisions* σ' | σ'. σ' $\in$ σ*-set*}) − {$\emptyset$}
$\longrightarrow$ ($\lambda c.$ $\neg$ ($\forall$ *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* $(\bigcup \sigma\text{-}set)$
and $\sigma \in \sigma\text{-}set \wedge p\text{-}set \in Pow\ (\bigcup\ \{consensus\text{-}value\text{-}property\text{-}decisions\ \sigma' \mid \sigma'.\ \sigma' \in \sigma\text{-}set\}) - \{\emptyset\}$
and $(\lambda c.\ \neg\ (\forall\ p \in p\text{-}set.\ p\ c)) \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma$
hence $\exists\ \sigma.\ \sigma \in \Sigma t \wedge \sigma \in \bigcap\ \{futures\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\}$
using *n-party-common-futures-exists* **by** *meson*
then obtain σ' **where** $\sigma' \in \Sigma t \wedge \sigma' \in \bigcap\ \{futures\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\}$ **by** *auto*
hence $\forall\ p \in p\text{-}set.\ \exists\ \sigma'' \in \sigma\text{-}set.\ state\text{-}property\text{-}is\text{-}decided\ (naturally\text{-}corresponding\text{-}state\text{-}property\ p,\ \sigma'')$
using $\langle\sigma \in \sigma\text{-}set \wedge p\text{-}set \in Pow\ (\bigcup\ \{consensus\text{-}value\text{-}property\text{-}decisions\ \sigma' \mid \sigma'.\ \sigma' \in \sigma\text{-}set\}) - \{\emptyset\}\rangle$
apply (*simp add*: *consensus-value-property-decisions-def consensus-value-property-is-decided-def*)
by *blast*
have $\forall\ \sigma'' \in \sigma\text{-}set.\ \sigma' \in futures\ \sigma''$
using $\langle\sigma' \in \Sigma t \wedge \sigma' \in \bigcap\text{-}Collect\ (futures\ \sigma)\ (\sigma \in \sigma\text{-}set)\rangle$ **by** *blast*
hence $\forall\ p \in p\text{-}set.\ state\text{-}property\text{-}is\text{-}decided\ (naturally\text{-}corresponding\text{-}state\text{-}property\ p,\ \sigma')$
using *forward-consistency* $\langle\forall\ p \in p\text{-}set.\ \exists\ \sigma'' \in \sigma\text{-}set.\ state\text{-}property\text{-}is\text{-}decided\ (naturally\text{-}corresponding\text{-}state\text{-}property\ p,\ \sigma'')\rangle$
by (*meson* $\langle\sigma' \in \Sigma t \wedge \sigma' \in \bigcap\text{-}Collect\ (futures\ \sigma)\ (\sigma \in \sigma\text{-}set)\rangle$ $\langle\sigma\text{-}set \subseteq \Sigma t\rangle$ *subsetCE*)
hence $state\text{-}property\text{-}is\text{-}decided\ (naturally\text{-}corresponding\text{-}state\text{-}property\ (\lambda c.\ \forall\ p \in p\text{-}set.\ p\ c),\ \sigma')$
apply (*simp add*: *naturally-corresponding-state-property-def state-property-is-decided-def*)
by *auto*
then show *False*
using $\langle(\lambda c.\ \neg\ (\forall\ p \in p\text{-}set.\ p\ c)) \in consensus\text{-}value\text{-}property\text{-}decisions\ \sigma\rangle$
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 $\langle\sigma \in \sigma\text{-}set \wedge p\text{-}set \in Pow\ (\bigcup\text{-}Collect\ (consensus\text{-}value\text{-}property\text{-}decisions\ \sigma')\ (\sigma' \in \sigma\text{-}set)) - \{\emptyset\}\rangle$ $\langle\sigma' \in \Sigma t \wedge \sigma' \in \bigcap\text{-}Collect\ (futures\ \sigma)\ (\sigma \in \sigma\text{-}set)\rangle$ *estimates-are-non-empty monotonic-futures* **by** *fastforce*
qed

end
theory *SafetyOracle*

imports *Main CBCCasper LatestMessage StateTransition ConsensusSafety*

begin

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

definition *is-agreeing* :: (*consensus-value-property* $*$ *state* $*$ *validator*) $\Rightarrow$ *bool*
where
is-agreeing $= (\lambda(p, \sigma, v).\ \forall\ c \in$ *latest-estimates-from-non-equivocating-validators* $\sigma\ v.\ p\ c)$

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*)

definition *disagreeing-validators* :: (*consensus-value-property* $*$ *state*) $\Rightarrow$ *validator set*
where
disagreeing-validators $= (\lambda(p, \sigma).\ \{v \in$ *observed-non-equivocating-validators* $\sigma.\ \exists\ c \in$ *latest-estimates-from-non-equivocating-validators* $\sigma\ v.\ \neg\ p\ c\})$

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

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

definition (**in** *Protocol*) *is-majority-driven* :: *consensus-value-property* ⇒ *bool*
where
is-majority-driven p = (∀ σ c. σ ∈ Σ ∧ c ∈ C ∧ *is-majority* (*agreeing-validators* (p, σ), σ) ⟶ (∀ c ∈ ε σ. p c))

definition (**in** *Protocol*) *is-max-driven* :: *consensus-value-property* ⇒ *bool*
where
is-max-driven p =
(∀ σ c. σ ∈ Σ ∧ c ∈ C ∧ *weight-measure* (*agreeing-validators* (p, σ)) > *weight-measure* (*disagreeing-validators* (p, σ)) ⟶ c ∈ ε σ ∧ p c)

definition *later-disagreeing-messages* :: (*consensus-value-property* ∗ *message* ∗ *validator* ∗ *state*) ⇒ *message set*
where
later-disagreeing-messages = (λ(p, m, v, σ).{m′ ∈ *later-from* (m, v, σ). ¬ p (*est* m′)})

lemma (**in** *Protocol*) *later-disagreeing-messages-type* :
∀ p σ v m. σ ∈ Σ ∧ v ∈ V ∧ m ∈ M ⟶ *later-disagreeing-messages* (p, m, v, σ) ⊆ M
unfolding *later-disagreeing-messages-def*
using *later-from-type-for-state* **by** *auto*

definition *is-clique* :: (*validator set* ∗ *consensus-value-property* ∗ *state*) ⇒ *bool*
where
is-clique = (λ(*v-set*, p, σ). (∀ v ∈ *v-set*. v ∈ *observed-non-equivocating-validators* σ
∧ (∀ v′ ∈ *v-set*.
is-singleton (*latest-messages-from-non-equivocating-validators*
(*the-elem* (*latest-justifications-from-non-equivocating-validators* σ v)) v′)
∧ *is-agreeing* (p, (*the-elem* (*latest-justifications-from-non-equivocating-validators* σ v)), v′)
∧ *later-disagreeing-messages* (p,
the-elem (*latest-messages-from-non-equivocating-validators*
(*the-elem* (*latest-justifications-from-non-equivocating-validators* σ v)) v′)

, v', σ) = $\emptyset$)))

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

have *later-from* $(m,v,\sigma) = \{m'' \in \sigma.$ *sender* $m'' = v\ \wedge$ *justified* $m\ m''\}$
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, σ) = *later-from* (m, v, σ')
using ‹$\{m'' \in \sigma \cup \{m'\}$. *sender* $m'' = v \wedge$ *justified* $m\ m''\} = \{m'' \in \sigma'$. *sender* $m'' = v \wedge$ *justified* $m\ m''\}$› *calculation* **by** *auto*
qed

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

lemma (**in** *Protocol*) *latest-justificationss-of-non-sender-not-affected-by-minimal-transitions*
:
$\forall\ \sigma\ \sigma'\ m'\ v.\ (\sigma, \sigma') \in$ *minimal-transitions*
$\longrightarrow m' =$ *the-elem* $(\sigma' - \sigma)$
$\longrightarrow v \in V - \{$*sender* $m'\}$
$\longrightarrow$ *latest-justifications-from-non-equivocating-validators* $\sigma\ v$ = *latest-justifications-from-non-equivocating-vali*
$\sigma'\ v$
oops

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

lemma (**in** *Protocol*) *clique-not-affected-by-minimal-transitions-outside-clique* :
$\forall\ \sigma\ \sigma'\ m'\ \textit{v-set}.\ (\sigma,\ \sigma') \in \textit{minimal-transitions} \wedge \textit{v-set} \subseteq V$
$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow \textit{is-clique}\ (\textit{v-set},\ p,\ \sigma) = \textit{is-clique}\ (\textit{v-set},\ p,\ \sigma')$
oops

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

lemma (**in** *Protocol*) *later-messages-from-non-equivocating-validator-include-all-earlier-messages*
:
$\forall\ v\ \sigma\ \sigma 1\ \sigma 2.\ \sigma \in \Sigma \wedge \sigma 1 \in \Sigma \wedge \sigma 1 \subseteq \sigma \wedge \sigma 2 \subseteq \sigma \wedge \sigma 1 \cap \sigma 2 = \emptyset$
$\longrightarrow (\forall\ m1 \in \sigma 1.\ \textit{sender}(m1) = v \longrightarrow (\forall\ m2 \in \sigma 2.\ \textit{sender}(m2) = v \longrightarrow m1$
$\in \textit{justification}(m2)))$
oops

lemma (**in** *Protocol*) *message-between-minimal-transition-is-latest-message* :
$\forall\ \sigma\ \sigma'\ m'\ v.\ (\sigma,\ \sigma') \in \textit{minimal-transitions}$
$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow v \notin \textit{equivocating-validators}\ \sigma'$
$\longrightarrow m' = \textit{the-elem}\ (\textit{latest-messages-from-non-equivocating-validators}\ \sigma'\ v)$
oops

lemma (**in** *Protocol*) *latest-message-from-non-equivocating-validator-is-previous-latest-or-later*:
$\forall\ \sigma\ \sigma'\ m'\ v.\ (\sigma,\ \sigma') \in \textit{minimal-transitions}$
$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow \textit{sender}\ m' \notin \textit{equivocating-validators}\ \sigma \wedge v \notin \textit{equivocating-validators}\ \sigma'$
$\longrightarrow \textit{the-elem}\ (\textit{latest-messages-from-non-equivocating-validators}\ (\textit{justification}\ m')$
$v)$
$= \textit{the-elem}\ (\textit{latest-messages-from-non-equivocating-validators}\ (\textit{the-elem}\ (\textit{latest-justifications-from-non-equi}$
$\sigma\ (\textit{sender}\ m')))\ v)$
$\vee\ \textit{justified}\ (\textit{the-elem}\ (\textit{latest-messages-from-non-equivocating-validators}\ (\textit{the-elem}$
$(\textit{latest-justifications-from-non-equivocating-validators}\ \sigma\ (\textit{sender}\ m')))\ v))$
$(\textit{the-elem}\ (\textit{latest-messages-from-non-equivocating-validators}$
$(\textit{justification}\ m')\ v))$
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-from* $(m1,\ sender\ m1,\ \sigma)$
apply (*simp add*: *later-from-def later-def from-sender-def*)
oops

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

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

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

lemma (**in** *Protocol*) *latest-message-justified-by-new-message* :
$\forall\ \sigma\ \sigma'\ m'\ v\text{-}set.\ (\sigma,\ \sigma') \in minimal\text{-}transitions \wedge v\text{-}set \subseteq V$

$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow \textit{sender}\ m' \notin \textit{equivocating-validators}\ \sigma'$
$\longrightarrow \textit{justified}\ (\textit{the-elem}\ (\textit{latest-messages-from-non-equivocating-validators}\ \sigma\ (\textit{sender}\ m')))\ m'$
oops

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

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

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

lemma (**in** *Protocol*) *empty-later-disagreeing-messages-in-new-message* :
$\forall\ \sigma\ \sigma'\ m'\ \textit{v-set}\ v\ p.\ (\sigma, \sigma') \in \textit{minimal-transitions} \land \textit{v-set} \subseteq V \land v \in V$
$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow \textit{sender}\ m' \notin \textit{equivocating-validators}\ \sigma'$
$\longrightarrow v \notin \textit{equivocating-validators}\ \sigma$
$\longrightarrow \textit{later-disagreeing-messages}\ (p,\ (\textit{the-elem}\ (\textit{latest-messages-from-non-equivocating-validators}\ (\textit{the-elem}\ (\textit{latest-justifications-from-non-equivocating-validators}\ \sigma\ (\textit{sender}\ m')))\ v)),\ v,\ \sigma) = \emptyset$
$\longrightarrow \textit{later-disagreeing-messages}\ (p,\ (\textit{the-elem}\ (\textit{latest-messages-from-non-equivocating-validators}\ (\textit{justification}\ m')\ v)),\ v,\ \sigma) = \emptyset$
oops

lemma (**in** *Protocol*) *clique-not-affected-by-minimal-transitions-outside-clique* :
$\forall\ \sigma\ \sigma'\ m'\ \textit{v-set}\ p.\ (\sigma, \sigma') \in \textit{minimal-transitions} \land \textit{v-set} \subseteq V$
$\longrightarrow \textit{is-majority-driven}\ p$
$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$

$\longrightarrow$ *is-clique* (*v-set*, *p*, σ) $\wedge$ *sender* m' $\in$ *v-set* $\wedge$ *sender* m' $\notin$ *equivocating-validators* σ'
$\wedge$ ($\forall$ $v \in$ *v-set*. *is-majority* (*v-set*, *the-elem* (*latest-justifications-from-non-equivocating-validators* σ v)))
$\longrightarrow$ *is-clique* (*v-set*, *p*, σ')
oops

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

lemma (**in** *Protocol*) *gt-threshold-imps-majority-for-any-validator* :
$\forall$ σ *v-set* p. $\sigma \in \Sigma$ $\wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *gt-threshold* (*v-set*, σ)
$\longrightarrow$ ($\forall$ $v \in$ *v-set*. *is-majority* (*v-set*, *the-elem* (*latest-justifications-from-non-equivocating-validators* σ v)))
oops

definition (**in** *Params*) *is-clique-oracle* :: (*validator set* $*$ *state* $*$ *consensus-value-property*) $\Rightarrow$ *bool*
where
is-clique-oracle
= (λ(*v-set*, σ, p). (*is-clique* (*v-set* $-$ (*equivocating-validators* σ), p, σ) $\wedge$ *gt-threshold* (*v-set* $-$ (*equivocating-validators* σ), σ)))

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

lemma (**in** *Protocol*) *clique-oracles-preserved-over-minimal-transitions-from-non-equivocating-validator* :
$\forall$ σ σ' m' *v-set* p. (σ, σ') $\in$ *minimal-transitions* $\wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *is-majority-driven* p

$\longrightarrow m' = \textit{the-elem}\ (\sigma' - \sigma)$
$\longrightarrow \textit{sender}\ m' \in \textit{v-set} - \textit{equivocating-validators}\ \sigma \wedge \textit{sender}\ m' \notin \textit{equivocating-validators}$
σ'
$\wedge\ \textit{is-clique-oracle}\ (\textit{v-set}, \sigma, p)$
$\longrightarrow \textit{is-clique-oracle}\ (\textit{v-set}, \sigma', p)$
oops

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

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

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

lemma (**in** *Protocol*) *clique-imps-everyone-agreeing* :
$\forall\ \sigma\ \textit{v-set}\ p.\ \sigma \in \Sigma \wedge \textit{v-set} \subseteq V$
$\longrightarrow \textit{is-clique}\ (\textit{v-set}, p, \sigma)$
$\longrightarrow \textit{v-set} \subseteq \textit{agreeing-validators}\ (p, \sigma)$
apply (*rule*, *rule*, *rule*, *rule*, *rule*)
proof−
fix σ *v-set* p **assume** $\sigma \in \Sigma \wedge \textit{v-set} \subseteq V$ **and** *is-clique* (*v-set*, p, σ)
then have *clique*: $\forall\ v \in \textit{v-set}.\ v \in \textit{observed-non-equivocating-validators}\ \sigma$
$\wedge\ \textit{is-singleton}\ (\textit{latest-messages-from-non-equivocating-validators}$
$(\textit{the-elem}\ (\textit{latest-justifications-from-non-equivocating-validators}$

σ v)) v)
$\wedge$ *later-disagreeing-messages* (p,
the-elem (*latest-messages-from-non-equivocating-validators*

(*the-elem* (*latest-justifications-from-non-equivocating-validators*
σ v)) v)
, v, σ) = $\emptyset$
by (*simp add*: *is-clique-def*)
then have *p-on-est* : $\forall$ $v \in$ *v-set*. ($\forall$ $m \in \{m' \in \sigma$. *sender* $m' = v$
$\wedge$ *justified* (*the-elem* (*latest-messages-from-non-equivocating-validators*

(*the-elem*
(*latest-justifications-from-non-equivocating-validators* σ v)) v))
$m'\}$.
p(*est* m))
by (*simp add*: *later-disagreeing-messages-def later-from-def later-def from-sender-def*)
have $\forall$ $v \in$ *v-set*. $v \in$ *observed-non-equivocating-validators* σ
using *clique* **by** *simp*
then have $\forall$ $v \in$ *v-set*. *the-elem* (*latest-justifications-from-non-equivocating-validators*
σ v)
= *justification* (*the-elem* (*latest-messages-from-non-equivocating-validators*
σ v))
apply (*simp add*: *latest-justifications-from-non-equivocating-validators-def*)
by (*metis* ‹$\sigma \in \Sigma \wedge$ *v-set* $\subseteq V$› *empty-iff is-singleton-the-elem observed-non-equivocating-validator-has-one-la*
singletonD singletonI the-elem-image-unique)
then have *justified-ok*: $\forall$ $v \in$ *v-set*. *justified* (*the-elem* (*latest-messages-from-non-equivocating-validators*

(*the-elem*
(*latest-justifications-from-non-equivocating-validators* σ v)) v))
(*the-elem* (*latest-messages-from-non-equivocating-validators*
σ v))
by (*smt* ‹$\sigma \in \Sigma \wedge$ *v-set* $\subseteq V$› *clique empty-iff is-singleton-the-elem justified-def*
latest-justifications-from-non-equivocating-validators-type latest-messages-from-non-equivocating-validators-def
latest-messages-is-subset-of-the-state observed-non-equivocating-validator-has-one-latest-justification
singletonI subsetCE)
have *sender-ok*: $\forall$ $v \in$ *v-set*. *sender* (*the-elem* (*latest-messages-from-non-equivocating-validators*
σ v)) = v
using ‹$\forall$ $v \in$ *v-set*. $v \in$ *observed-non-equivocating-validators* σ› *sender-of-latest-message-of-observed-non-equ*
using ‹$\sigma \in \Sigma \wedge$ *v-set* $\subseteq V$› **by** *blast*
have $\forall$ $v \in$ *v-set*. *the-elem* (*latest-messages-from-non-equivocating-validators* σ
v) $\in \sigma$
using ‹$\forall$ $v \in$ *v-set*. $v \in$ *observed-non-equivocating-validators* σ› *latest-message-of-observed-non-equivocating-*
using ‹$\sigma \in \Sigma \wedge$ *v-set* $\subseteq V$› **by** *blast*
then have $\forall$ $v \in$ *v-set*. p (*est* (*the-elem* (*latest-messages-from-non-equivocating-validators*
σ v)))
using *p-on-est sender-ok justified-ok*
by *blast*
then have $\forall$ $v \in$ *v-set*. p (*the-elem* (*latest-estimates-from-non-equivocating-validators*
σ v))

apply (*simp add*: *latest-estimates-from-non-equivocating-validators-def*)
by (*metis* (*no-types*, *lifting*) ‹$\forall$ *v*$\in$*v-set*. *v* $\in$ *observed-non-equivocating-validators* σ› ‹$\sigma \in \Sigma \land$ *v-set* $\subseteq V$› *empty-iff is-singleton-the-elem observed-non-equivocating-validator-has-one-latest-messa* *singletonD singletonI the-elem-image-unique*)
then show *v-set* $\subseteq$ *agreeing-validators* (p, σ)
unfolding *agreeing-validators-def*
by (*smt* ‹$\forall$ *v*$\in$*v-set*. *v* $\in$ *observed-non-equivocating-validators* σ› ‹$\sigma \in \Sigma \land$ *v-set* $\subseteq V$› *is-singleton-the-elem mem-Collect-eq observed-non-equivocating-validator-has-one-latest-estimate old.prod.case singletonD subsetI*)
qed

lemma (**in** *Protocol*) *threshold-sized-clique-imps-estimator-agreeing* :
$\forall$ σ *v-set* p. $\sigma \in \Sigma t \land$ *v-set* $\subseteq V$
$\longrightarrow$ *finite v-set*
$\longrightarrow$ *is-majority-driven* p
$\longrightarrow$ *is-clique* (*v-set* $-$ *equivocating-validators* σ, p, σ) $\land$ *gt-threshold* (*v-set* $-$ *equivocating-validators* σ, σ)
$\longrightarrow$ ($\forall$ $c \in \varepsilon$ σ. p c)
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof $-$
fix σ *v-set* p c
assume $\sigma \in \Sigma t \land$ *v-set* $\subseteq V$
and *finite v-set*
and *is-majority-driven* p
and *is-clique* (*v-set* $-$ *equivocating-validators* σ, p, σ) $\land$ *gt-threshold* (*v-set* $-$ *equivocating-validators* σ, σ)
and $c \in \varepsilon$ σ
then have *v-set* $-$ *equivocating-validators* σ $\subseteq$ *agreeing-validators* (p, σ)
using *clique-imps-everyone-agreeing*
by (*meson Diff-subset* Σ*t-is-subset-of-*Σ *subsetCE subset-trans*)
then have *weight-measure* (*v-set* $-$ *equivocating-validators* σ) $\leq$ *weight-measure* (*agreeing-validators* (p, σ))
using *agreeing-validators-finite equivocating-validators-def weight-measure-comparison-strict-subset-gte* Σ*t-is-subset-of-*Σ ‹$\sigma \in \Sigma t \land$ *v-set* $\subseteq V$› ‹*finite v-set*› **by** *auto*
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) $\land$ *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 Σ*t-def* ‹$\sigma \in \Sigma t \land$ *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-comparison-strict-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 ⟨*is-majority-driven* p⟩ **unfolding** *is-majority-driven-def is-majority-def gt-threshold-def*
using ⟨$c \in \varepsilon\ \sigma$⟩
using *Mi.simps* Σ*t-is-subset-of-*Σ ⟨$\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$⟩ *non-justifying-message-exists-in-M-0* **by** *blast*
qed

lemma (**in** *Protocol*) *clique-oracle-for-all-futures* :
$\forall$ σ *v-set* p. $\sigma \in \Sigma t \wedge$ *v-set* $\subseteq V$
$\longrightarrow$ *is-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** *is-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*
$\longrightarrow$ *is-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

type-synonym *block* = *consensus-value*

locale *BlockchainParams* = *Params* +

fixes *B* :: *block set*
fixes *genesis* :: *block*

and *prev* :: *block* ⇒ *block*

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

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

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

definition (**in** *BlockchainParams*) *score* :: *state* ⇒ *block* ⇒ *real*
where
score σ *b* = *sum W* {*v* ∈ *observed* σ. ∃ *b*′ ∈ *B*. *b*′ ∈ (*latest-estimates-from-non-equivocating-validators* σ *v*) ∧ (*b* ⇂ *b*′)}

definition (**in** *BlockchainParams*) *children* :: *block* ∗ *state* ⇒ *block set*
where

children = (λ(*b*, σ). {*b'* $\in$ *est* 'σ. *b* = *prev* *b'*})

definition (**in** *BlockchainParams*) *best-children* :: *block* $*$ *state* $\Rightarrow$ *block set*
where
best-children = (λ (*b*, σ). {*arg-max-on* (*score* σ) (*children* (*b*, σ))})

function (**in** *BlockchainParams*) *GHOST* :: (*block set* $*$ *state*) => *block 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-estimator* :: *state* $\Rightarrow$ *block set*
where
GHOST-estimator σ = *GHOST* ({*genesis*}, σ) $\cup$ ($\bigcup$ *b* $\in$ *GHOST* ({*genesis*}, σ). *children* (*b*, σ))

abbreviation (**in** *BlockchainParams*) *P* :: *consensus-value-property set*
where
P $\equiv$ {*p*. $\exists$!*b* $\in$ *B*. $\forall$ *b'* $\in$ *B*. (*b* $\downharpoonleft$ *b'* $\longrightarrow$ *p* *b'* = *True*) $\wedge$ $\neg$ (*b* $\downharpoonleft$ *b'* $\longrightarrow$ *p* *b'* = *False*)}

locale *Blockchain* = *BlockchainParams* + *Protocol* +
assumes *blockchain-type* : $\forall$ *b* *b'* *b''*. {*b*, *b'*, *b''*} $\subseteq$ *B* $\longrightarrow$ *b'* $\downharpoonleft$ *b* $\wedge$ *b''* $\downharpoonleft$ *b* $\longrightarrow$ (*b'* $\downharpoonleft$ *b''* $\vee$ *b''* $\downharpoonleft$ *b'*)
and *block-is-consensus-value* : *B* = *C*

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

definition (**in** *BlockchainParams*) *block-conflicting* :: (*block* $*$ *block*) $\Rightarrow$ *bool*
where
block-conflicting = (λ(*b1*, *b2*). $\neg$ (*b1* $\downharpoonleft$ *b2* $\vee$ *b2* $\downharpoonleft$ *b1*))

lemma (**in** *Blockchain*) *conflicting-blocks-imps-conflicting-decision* :
$\forall$ *b1* *b2* σ. {*b1*, *b2*} $\subseteq$ *B* $\wedge$ σ $\in$ Σ
$\longrightarrow$ *block-conflicting* (*b1*, *b2*)
$\longrightarrow$ *consensus-value-property-is-decided* (*block-membership-property* *b1*, σ)
$\longrightarrow$ *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership-property* *b2*), σ)
apply (*simp add*: *block-membership-property-def* *consensus-value-property-is-decided-def*

naturally-corresponding-state-property-def state-property-is-decided-def)
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix *b1 b2* σ
assume *b1* $\in B \wedge$ *b2* $\in B \wedge \sigma \in \Sigma$ **and** *block-conflicting* (*b1*, *b2*) **and** $\forall \sigma \in$*futures* σ. $\forall b' \in \varepsilon\ \sigma$. *b1* $\downharpoonleft b'$
show $\forall \sigma \in$*futures* σ. $\forall c \in \varepsilon\ \sigma$. $\neg$ *b2* $\downharpoonleft c$
proof (*rule ccontr*)
assume $\neg$ ($\forall \sigma \in$*futures* σ. $\forall c \in \varepsilon\ \sigma$. $\neg$ *b2* $\downharpoonleft c$)
hence $\exists\ \sigma\ \in$*futures* σ. $\exists\ c \in \varepsilon\ \sigma$. *b2* $\downharpoonleft c$
by *blast*
hence $\exists\ \sigma\ \in$*futures* σ. $\exists\ c \in \varepsilon\ \sigma$. *b2* $\downharpoonleft c \wedge$ *b1* $\downharpoonleft c$
using ‹$\forall \sigma \in$*futures* σ. $\forall b' \in \varepsilon\ \sigma$. *b1* $\downharpoonleft b'$› **by** *simp*
hence *b1* $\downharpoonleft$ *b2* $\vee$ *b2* $\downharpoonleft$ *b1*
using *blockchain-type*
apply (*simp*)
using Σ*t-is-subset-of-*Σ ‹*b1* $\in B \wedge$ *b2* $\in B \wedge \sigma \in \Sigma$› *block-is-consensus-value 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* :
$\forall$ σ*-set*. σ*-set* $\subseteq \Sigma t$
$\longrightarrow$ *finite* σ*-set*
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ σ*-set*)
$\longrightarrow$ ($\forall$ σ σ' *b1 b2*. $\{\sigma, \sigma'\} \subseteq \sigma$*-set* $\wedge$ {*b1*, *b2*} $\subseteq B \wedge$ *block-conflicting* (*b1*, *b2*) $\wedge$ *block-membership-property b1* $\in$ *consensus-value-property-decisions* σ
$\longrightarrow$ *block-membership-property b2* $\notin$ *consensus-value-property-decisions* σ')
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix σ*-set* σ σ' *b1 b2*
assume σ*-set* $\subseteq \Sigma t$ **and** *finite* σ*-set* **and** *is-faults-lt-threshold* ($\bigcup \sigma$*-set*)
and $\{\sigma, \sigma'\} \subseteq \sigma$*-set* $\wedge$ {*b1*, *b2*} $\subseteq B \wedge$ *block-conflicting* (*b1*, *b2*) $\wedge$ *block-membership-property b1* $\in$ *consensus-value-property-decisions* σ
and *block-membership-property b2* $\in$ *consensus-value-property-decisions* σ'
hence $\neg$ *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership-property b1*), σ')
using *negation-is-not-decided-by-other-validator* ‹σ*-set* $\subseteq \Sigma t$› ‹*finite* σ*-set*› ‹*is-faults-lt-threshold* ($\bigcup \sigma$*-set*)› **apply** (*simp add*: *consensus-value-property-decisions-def*)

using ‹$\{\sigma, \sigma'\} \subseteq \sigma$*-set* $\wedge$ {*b1*, *b2*} $\subseteq B \wedge$ *block-conflicting* (*b1*, *b2*) $\wedge$ *block-membership-property b1* $\in$ *consensus-value-property-decisions* σ› **by** *auto*
have {*b1*, *b2*} $\subseteq B \wedge \sigma \in \Sigma \wedge$ *block-conflicting* (*b1*, *b2*)
using Σ*t-is-subset-of-*Σ ‹σ*-set* $\subseteq \Sigma t$› ‹$\{\sigma, \sigma'\} \subseteq \sigma$*-set* $\wedge$ {*b1*, *b2*} $\subseteq B \wedge$ *block-conflicting* (*b1*, *b2*) $\wedge$ *block-membership-property b1* $\in$ *consensus-value-property-decisions* σ› **by** *auto*

hence *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership-property b1*), σ')
using ‹*block-membership-property b2* ∈ *consensus-value-property-decisions* σ'› *conflicting-blocks-imps-conflicting-decision*
apply (*simp add*: *consensus-value-property-decisions-def*)
by (*metis* ‹σ*-set* ⊆ Σ*t*› ‹*finite* σ*-set*› ‹*is-faults-lt-threshold* ($\bigcup \sigma$*-set*)› ‹{σ, σ'} ⊆ σ*-set* ∧ {*b1*, *b2*} ⊆ *B* ∧ *block-conflicting* (*b1*, *b2*) ∧ *block-membership-property b1* ∈ *consensus-value-property-decisions* σ› *conflicting-blocks-imps-conflicting-decision consensus-value-property-decisions-def insert-subset mem-Collect-eq negation-is-not-decided-by-other-validator*)

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

theorem (**in** *Blockchain*) *no-decision-on-conflicting-blocks* :
∀ $\sigma 1$ $\sigma 2$. {$\sigma 1$, $\sigma 2$} ⊆ Σ*t*
⟶ *is-faults-lt-threshold* ($\sigma 1$ ∪ $\sigma 2$)
⟶ (∀ *b1 b2*. {*b1*, *b2*} ⊆ *C* ∧ *block-conflicting* (*b1*, *b2*)
⟶ *block-membership-property b1* ∈ *consensus-value-property-decisions* $\sigma 1$
⟶ *block-membership-property b2* ∉ *consensus-value-property-decisions* $\sigma 2$)
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)
proof −
fix $\sigma 1$ $\sigma 2$ *b1 b2*
assume {$\sigma 1$, $\sigma 2$} ⊆ Σ*t* **and** *is-faults-lt-threshold* ($\sigma 1$ ∪ $\sigma 2$) **and** {*b1*, *b2*} ⊆ *C* ∧ *block-conflicting* (*b1*, *b2*)
and *block-membership-property b1* ∈ *consensus-value-property-decisions* $\sigma 1$
and *block-membership-property b2* ∈ *consensus-value-property-decisions* $\sigma 2$
hence *consensus-value-property-is-decided* (*block-membership-property b1*, $\sigma 1$)
by (*simp add*: *consensus-value-property-decisions-def*)
hence ¬ *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership-property b1*), $\sigma 2$)
using *two-party-consensus-safety-for-consensus-value-property* ‹*is-faults-lt-threshold* ($\sigma 1$ ∪ $\sigma 2$)› ‹{$\sigma 1$, $\sigma 2$} ⊆ Σ*t*› **by** *blast*
have *block-membership-property b2* ∈ *consensus-value-property-decisions* $\sigma 2$
using ‹*block-membership-property b2* ∈ *consensus-value-property-decisions* $\sigma 2$›

by (*simp add*: *consensus-value-property-decisions-def*)
have $\sigma 2$ ∈ Σ*t* ∧ {*b2*, *b1*} ⊆ *B* ∧ *block-conflicting* (*b2*, *b1*)
using *block-is-consensus-value* ‹{$\sigma 1$, $\sigma 2$} ⊆ Σ*t*› ‹{*b1*, *b2*} ⊆ *C* ∧ *block-conflicting* (*b1*, *b2*)› **by** (*simp add*: *block-conflicting-def*)
hence *consensus-value-property-is-decided* (*consensus-value-property-not* (*block-membership-property b1*), $\sigma 2$)
using *conflicting-blocks-imps-conflicting-decision* ‹*block-membership-property b2* ∈ *consensus-value-property-decisions* $\sigma 2$›
using Σ*t-is-subset-of*-Σ *consensus-value-property-decisions-def* **by** *auto*
then show *False*
using ‹¬ *consensus-value-property-is-decided* (*consensus-value-property-not*

(*block-membership-property b1*), *σ2*)› **by** *blast*
qed

locale *Ghost* = *BlockchainParams* + *Protocol* +
assumes *block-type* : $\forall\ b.\ b \in B \longleftrightarrow prev\ b \in B$
and *block-is-consensus-value* : $B = C$
and *ghost-is-estimator* : $\varepsilon = GHOST\text{-}estimator$
and *genesis-type* : $genesis \in C$

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

lemma *argmax-type* :
$S \subseteq A \Longrightarrow arg\text{-}max\text{-}on\ f\ S \in A$
apply (*simp add*: *arg-max-on-def arg-max-def is-arg-max-def*)
oops

lemma (**in** *Ghost*) *best-children-type* :
$\forall\ b\ \sigma.\ b \in B \wedge \sigma \in \Sigma \longrightarrow\ best\text{-}children\ (b, \sigma) \subseteq B$
apply (*simp add*: *best-children-def arg-max-on-def arg-max-def is-arg-max-def*)
using *children-type*
apply *auto*
oops

lemma (**in** *Ghost*) *GHSOT-type* :
$\forall\ \sigma\ b\text{-}set.\ \sigma \in \Sigma \wedge b\text{-}set \subseteq B \longrightarrow\ GHOST(b\text{-}set, \sigma) \subseteq B$
oops

lemma (**in** *BlockchainParams*) *GHOST-is-valid-estimator* :
$(\forall\ b.\ b \in B \longleftrightarrow prev\ b \in B) \wedge B = C \wedge genesis \in C$
$\Longrightarrow$ *is-valid-estimator GHOST-estimator*
apply (*simp add*: *is-valid-estimator-def BlockchainParams.GHOST-estimator-def*)
oops

lemma (**in** *Ghost*) *block-membership-property-is-majority-driven* :
$\forall\ p \in P.$ *is-majority-driven p*
apply (*simp add*: *is-majority-driven-def*)

oops

lemma (**in** *Ghost*) *block-membership-property-is-max-driven* :
$\forall\ p \in P.$ *is-max-driven p*
apply (*simp add*: *is-max-driven-def*)

oops

end