

Minimal CBC Casper Isabelle/HOL proofs

LayerX

March 17, 2019

Contents

1	Description of CBC Casper	4
2	Message Justification	11
3	Latest Message	15
4	Safety Proof	29

theory *Strict-Order*

imports *Main*

begin

notation *Set.empty* (\emptyset)

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

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

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

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

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

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

definition $\text{minimal-on} :: 'a \ \text{set} \Rightarrow 'a \ \text{rel} \Rightarrow 'a \Rightarrow \text{bool}$

where

$\text{minimal-on } A \ r \ x = (x \in A \wedge (\forall y. (y, x) \in r \longrightarrow y \notin A))$

definition $\text{maximal-on} :: 'a \ \text{set} \Rightarrow 'a \ \text{rel} \Rightarrow 'a \Rightarrow \text{bool}$

where

$\text{maximal-on } A \ r \ x = (x \in A \wedge (\forall y. (x, y) \in r \longrightarrow y \notin A))$

lemma $\text{maximal-and-maximum-coincide-for-strict-linear-order} :$

$\text{strict-linear-order-on } A \ r \Longrightarrow \text{maximal-on } A \ r \ x = \text{maximum-on } A \ r \ x$

apply ($\text{simp add: strict-linear-order-on-def irreft-def total-on-def trans-def maximal-on-def maximum-on-def upper-bound-on-def}$)

by blast

lemma $\text{strict-partial-order-on-finite-non-empty-set-has-maximal} :$

$\text{strict-partial-order } r \longrightarrow \text{finite } A \longrightarrow A \neq \emptyset \longrightarrow (\exists x. \text{maximal-on } A \ r \ x)$

proof –

have $\bigwedge n. \text{strict-partial-order } r \Longrightarrow (\forall A. \text{Suc } n = \text{card } A \longrightarrow \text{finite } A \longrightarrow A \neq \emptyset \longrightarrow (\exists x. \text{maximal-on } A \ r \ x))$

proof –

assume $\text{strict-partial-order } r$

then have $(\forall a. (a, a) \notin r)$

by ($\text{simp add: strict-partial-order-def irreft-def}$)

fix n

show $\forall A. \text{Suc } n = \text{card } A \longrightarrow \text{finite } A \longrightarrow A \neq \emptyset \longrightarrow (\exists x. \text{maximal-on } A \ r \ x)$

apply ($\text{induction } n$)

unfolding maximal-on-def

using $\langle (\forall a. (a, a) \notin r) \rangle$

apply ($\text{metis card-eq-SucD empty-iff insert-iff}$)

proof –

fix n

assume $\forall A. \text{Suc } n = \text{card } A \longrightarrow \text{finite } A \longrightarrow A \neq \emptyset \longrightarrow (\exists x. x \in A \wedge (\forall y. (x, y) \in r \longrightarrow y \notin A))$

have $\forall B. \text{Suc } (\text{Suc } n) = \text{card } B \longrightarrow \text{finite } B \longrightarrow B \neq \emptyset \longrightarrow (\exists A' b. B = A' \cup \{b\} \wedge \text{card } A' = \text{Suc } n \wedge b \notin A')$

by ($\text{metis Un-commute add-diff-cancel-left' card-gt-0-iff card-insert-disjoint card-le-Suc-iff insert-is-Un not-le not-less-eq-eq plus-1-eq-Suc}$)

then have $\forall B. \text{Suc } (\text{Suc } n) = \text{card } B \longrightarrow \text{finite } B \longrightarrow B \neq \emptyset \longrightarrow (\exists A' b. B = A' \cup \{b\} \wedge \text{card } A' = \text{Suc } n \wedge \text{finite } A' \wedge A' \neq \emptyset \wedge b \notin A')$

by ($\text{metis card-gt-0-iff zero-less-Suc}$)

then have $\forall B. \text{Suc } (\text{Suc } n) = \text{card } B \longrightarrow \text{finite } B \longrightarrow B \neq \emptyset$

$\longrightarrow (\exists A' b x. B = A' \cup \{b\} \wedge b \notin A' \wedge x \in A' \wedge (\forall y. (x, y) \in r \longrightarrow y \notin A'))$

using $\langle \forall A. \text{Suc } n = \text{card } A \longrightarrow \text{finite } A \longrightarrow A \neq \emptyset \longrightarrow (\exists x. x \in A \wedge (\forall y. (x, y) \in r \longrightarrow y \notin A)) \rangle$

by metis

then show $\forall B. \text{Suc } (\text{Suc } n) = \text{card } B \longrightarrow \text{finite } B \longrightarrow B \neq \emptyset \longrightarrow (\exists x. x \in B \wedge (\forall y. (x, y) \in r \longrightarrow y \notin B))$
by (*metis* (*no-types*, *lifting*) *Un-insert-right* $\langle \forall a. (a, a) \notin r \rangle$ (*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* :

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

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

lemma *strict-linear-order-on-finite-non-empty-set-has-one-maximum* :

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

$\text{strict-partial-order-has-at-most-one-maximum}$ *maximal-and-maximum-coincide-for-strict-linear-order*
by *fastforce*

end

1 Description of CBC Casper

theory *CBCCasper*

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

begin

notation *Set.empty* (\emptyset)

typedecl *validator*

typedecl *consensus-value*

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

type-synonym *state* = *message set*

fun *sender* :: *message* \Rightarrow *validator*
 where
 sender (*Message* ($-, v, -$)) = *v*

fun *est* :: *message* \Rightarrow *consensus-value*
 where
 est (*Message* (*c*, $-, -$)) = *c*

fun *justification* :: *message* \Rightarrow *state*
 where
 justification (*Message* ($-, -, s$)) = *set s*

fun
 Σi :: (*validator set* \times *consensus-value set* \times (*message set* \Rightarrow *consensus-value set*)) \Rightarrow *nat* \Rightarrow *state set* **and**
 $M i$:: (*validator set* \times *consensus-value set* \times (*message set* \Rightarrow *consensus-value set*)) \Rightarrow *nat* \Rightarrow *message set*
 where

$\Sigma i \ (V, C, \varepsilon) \ 0 = \{\emptyset\}$
 $\mid \Sigma i \ (V, C, \varepsilon) \ n = \{\sigma \in \text{Pow} \ (Mi \ (V, C, \varepsilon) \ (n - 1)). \text{finite } \sigma \wedge (\forall \ m. \ m \in \sigma \longrightarrow \text{justification } m \subseteq \sigma)\}$
 $\mid Mi \ (V, C, \varepsilon) \ n = \{m. \text{est } m \in C \wedge \text{sender } m \in V \wedge \text{justification } m \in (\Sigma i \ (V, C, \varepsilon) \ n) \wedge \text{est } m \in \varepsilon \ (\text{justification } m)\}$

locale *Params* =
fixes $V :: \text{validator set}$
and $W :: \text{validator} \Rightarrow \text{real}$
and $t :: \text{real}$
fixes $C :: \text{consensus-value set}$
and $\varepsilon :: \text{message set} \Rightarrow \text{consensus-value set}$

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* :: $(\text{state} \Rightarrow \text{consensus-value set}) \Rightarrow \text{bool}$

where

is-valid-estimator $e = (\forall \sigma \in \Sigma. \ e \ \sigma \in \text{Pow } C - \{\emptyset\})$

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

by *force*

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

by *auto*

lemma *Mi -subset-to- Σi* : $Mi \ (V, C, \varepsilon) \ n \subseteq Mi \ (V, C, \varepsilon) \ (n+1) \Longrightarrow \Sigma i \ (V, C, \varepsilon) \ (n+1) \subseteq \Sigma i \ (V, C, \varepsilon) \ (n+2)$

by *auto*

lemma *Σi -monotonic*: $\Sigma i \ (V, C, \varepsilon) \ n \subseteq \Sigma i \ (V, C, \varepsilon) \ (n+1)$

apply *(induction n)*

apply *simp*

apply *(metis Mi-subset-to- Σi Suc-eq-plus1 Σi -subset-to- Mi add commute add-2-eq-Suc)*

done

lemma *Mi -monotonic*: $Mi \ (V, C, \varepsilon) \ n \subseteq Mi \ (V, C, \varepsilon) \ (n+1)$

apply *(induction n)*

defer

using *Σi -monotonic Σi -subset-to- Mi* **apply** *blast*

apply *auto*

done

lemma *Σi -monotonicity*: $\forall \ m \in \mathbb{N}. \ \forall \ n \in \mathbb{N}. \ m \leq n \longrightarrow \Sigma i \ (V, C, \varepsilon) \ m \subseteq \Sigma i \ (V, C, \varepsilon) \ n$

using *Σi -monotonic*

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

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

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

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

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

lemma *message-is-in-Mi-n* :
 $\forall m \in M. \exists n \in \mathbf{N}. m \in \text{Mi } (V, C, \varepsilon) n$
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 σm
assume $\sigma \in \Sigma \wedge m \in \sigma$
have
 $\exists n \in \mathbf{N}. m \in \text{Mi } (V, C, \varepsilon) n$

```

     $\implies m \in M$ 
    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  $\langle \sigma \in \Sigma \wedge m \in \sigma \rangle$  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 :  $\forall \sigma \in \Sigma. \text{finite } \sigma$ 
  apply (simp add:  $\Sigma\text{-def}$ )
  using Params.Σi-monotonic by fastforce

```

```

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

```

```

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

```

```

lemma M-type:  $\bigwedge m. m \in M \implies \text{est } m \in C \wedge \text{sender } m \in V \wedge \text{justification } m \in \Sigma$ 
  unfolding M-def Σ-def
  by auto

```

end

```

locale Protocol = Params +
  assumes V-type:  $V \neq \emptyset$ 
  and W-type:  $\bigwedge w. w \in \text{range } W \implies w > 0$ 
  and t-type:  $0 \leq t \ t < \text{Sum } (W \text{ ` } V)$ 
  and C-type:  $\text{card } C > 1$ 
  and ε-type: is-valid-estimator  $\varepsilon$ 

```

```

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

```

```

lemma (in Protocol) estimates-are-subset-of-C:  $\bigwedge \sigma. \sigma \in \Sigma \implies \varepsilon \sigma \subseteq C$ 
  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:  $\Sigma$ -def)
using Nats-0  $\Sigma$ i.simps(1) by blast

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

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

lemma (in Protocol) estimates-exists-for-empty-set :
 $\varepsilon \emptyset \neq \emptyset$ 
by (simp add: empty-set-exists-in- $\Sigma$  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- $\Sigma$   $\varepsilon$ -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)  $\Sigma$ i-is-subset-of- $\Sigma$  :
 $\forall n \in \mathbb{N}. \Sigma i \ (V, C, \varepsilon) \ n \subseteq \Sigma$ 
by (simp add:  $\Sigma$ -def SUP-upper)

lemma (in Protocol) message-justifying-state-in- $\Sigma$ -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  $\Sigma$ -is-subset-of- $\Sigma$  by auto
have  $\varepsilon \sigma \neq \emptyset$ 
  using estimates-are-non-empty  $\langle \sigma \in \Sigma \rangle$  by auto
have finite  $\sigma$ 
  using state-is-finite  $\langle \sigma \in \Sigma \rangle$  by auto
moreover have  $\exists m. \text{sender } m \in V \wedge \text{est } m \in \varepsilon \sigma \wedge \text{justification } m = \sigma$ 
  using est.simps sender.simps justification.simps  $V$ -type  $\langle \varepsilon \sigma \neq \emptyset \rangle \langle \text{finite } \sigma \rangle$ 
  by (metis all-not-in-conv finite-list)
moreover have  $\varepsilon \sigma \subseteq C$ 
  using estimates-are-subset-of- $C$   $\Sigma$ -is-subset-of- $\Sigma$   $\langle n \in \mathbb{N} \rangle \langle \sigma \in \Sigma_i (V, C, \varepsilon) \rangle$ 
  by blast
ultimately show  $\exists m. \text{est } m \in C \wedge \text{sender } m \in V \wedge \text{justification } m \in \Sigma_i (V, C, \varepsilon) n \wedge \text{est } m \in \varepsilon (\text{justification } m) \wedge \text{justification } m = \sigma$ 
  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: $\Sigma \subset \text{Pow } M$

proof –

```

obtain  $m$  where  $m \in \text{Mi } (V, C, \varepsilon) 0 \wedge \text{justification } m = \emptyset$ 
  using non-justifying-message-exists-in- $M$ -0 by auto
then have  $\{m\} \in \Sigma_i (V, C, \varepsilon) (\text{Suc } 0)$ 
  using Params. $\Sigma$ -subset- $\text{Mi}$  by auto
then have  $\exists m'. m' \in \text{Mi } (V, C, \varepsilon) (\text{Suc } 0) \wedge \text{justification } m' = \{m\}$ 
  using message-justifying-state-in- $\Sigma$ - $n$ -exists-in- $M$ - $n$  Nats-1 One-nat-def by
metis
then obtain  $m'$  where  $m' \in \text{Mi } (V, C, \varepsilon) (\text{Suc } 0) \wedge \text{justification } m' = \{m\}$ 
by auto
then have  $\{m'\} \in \text{Pow } M$ 
  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- $\text{Mi}$  Protocol-axioms  $\langle m' \in \text{Mi } (V, C, \varepsilon) (\text{Suc } 0) \wedge \text{justification } m' = \{m\} \rangle$ 
  by fastforce
ultimately show ?thesis
  using  $\Sigma$ -subsesteq-of-pow- $M$  by auto
qed

```

lemma (in Protocol) M -type-counterexample:

$(\forall \sigma. \varepsilon \sigma = C) \implies M = \{m. \text{est } m \in C \wedge \text{sender } m \in V \wedge \text{justification } m \in \Sigma\}$

```

apply (simp add:  $M$ -def)
apply auto
using  $\Sigma$ -is-subset-of- $\Sigma$  apply blast
by (simp add:  $\Sigma$ -def)

```

definition *observed* :: *message set* \Rightarrow *validator set*

where

observed $\sigma = \{ \text{sender } m \mid m. m \in \sigma \}$

lemma (in *Protocol*) *observed-type* :

$\forall \sigma \in \text{Pow } M. \text{observed } \sigma \in \text{Pow } V$

using *Params.M-type Protocol-axioms observed-def* **by** *fastforce*

lemma (in *Protocol*) *observed-type-for-state* :

$\forall \sigma \in \Sigma. \text{observed } \sigma \subseteq V$

using *Params.M-type Protocol-axioms observed-def state-is-subset-of-M* **by** *fastforce*

fun *is-future-state* :: (*state* * *state*) \Rightarrow *bool*

where

is-future-state ($\sigma 1, \sigma 2$) = ($\sigma 1 \subseteq \sigma 2$)

lemma (in *Params*) *state-difference-is-valid-message* :

$\forall \sigma \sigma'. \sigma \in \Sigma \wedge \sigma' \in \Sigma$

$\longrightarrow \text{is-future-state}(\sigma, \sigma')$

$\longrightarrow \sigma' - \sigma \subseteq M$

using *state-is-subset-of-M* **by** *blast*

definition *justified* :: *message* \Rightarrow *message* \Rightarrow *bool*

where

justified $m1 \ m2 = (m1 \in \text{justification } m2)$

definition *equivocation* :: (*message* * *message*) \Rightarrow *bool*

where

equivocation =

$(\lambda(m1, m2). \text{sender } m1 = \text{sender } m2 \wedge m1 \neq m2 \wedge \neg (\text{justified } m1 \ m2) \wedge \neg (\text{justified } m2 \ m1))$

definition *is-equivocating* :: *state* \Rightarrow *validator* \Rightarrow *bool*

where

is-equivocating $\sigma \ v = (\exists m1 \in \sigma. \exists m2 \in \sigma. \text{equivocation } (m1, m2) \wedge \text{sender } m1 = v)$

definition *equivocating-validators* :: *state* \Rightarrow *validator set*

where

equivocating-validators $\sigma = \{v \in \text{observed } \sigma. \text{is-equivocating } \sigma \ v\}$

lemma (in *Protocol*) *equivocating-validators-type* :

$\forall \sigma \in \Sigma. \text{equivocating-validators } \sigma \subseteq V$

using *observed-type-for-state equivocating-validators-def* **by** *blast*

definition (in *Params*) *equivocating-validators-paper* :: *state* \Rightarrow *validator set*

where
equivocating-validators-paper $\sigma = \{v \in V. \text{is-equivocating } \sigma \ v\}$

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

definition (*in Params*) *equivocation-fault-weight* :: *state* \Rightarrow *real*
where
equivocation-fault-weight $\sigma = \text{sum } W \ (\text{equivocating-validators } \sigma)$

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

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

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

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

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

end

2 Message Justification

theory *MessageJustification*

imports *Main CBCCaspar Libraries/LaTeXsugar*

begin

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

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 $\langle est\ m \in C \rangle \langle est\ m \in \varepsilon\ (justification\ m) \rangle \langle justification\ m \in \Sigma i\ (V, C, \varepsilon)\ n \rangle \langle m \in justification\ m \rangle \langle sender\ m \in V \rangle$ 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 $\langle m \in justification\ m \rangle$ 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 $\langle m \in justification\ m \rangle$ **by** blast
qed

lemma (in *Protocol*) *message-cannot-justify-itself* :
 $(\forall\ m \in M. \neg\ justified\ m\ m)$

proof –
have *irrefl message-justification*
using *irreflexivity-of-justifications* **by** simp
then show ?thesis
by (simp add: irreflexivity-of-justifications irrefl-def message-justification-def)
qed

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

by (*simp add: irreflexivity-of-justifications transitivity-of-justifications*)

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

lemma (*in Protocol*) *strict-monotonicity-of-justifications* :
 $\forall m m' \sigma. m \in M \wedge \sigma \in \Sigma \wedge \text{justified } m' m \longrightarrow \text{justification } m' \subset \text{justification } m$
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* :
 $\forall m m'. m \in M \wedge m' \in M \longrightarrow \text{justified } m' m \longrightarrow m \neq m'$
using *message-cannot-justify-itself* **by** *auto*

lemma (*in Protocol*) *only-valid-message-is-justified* :
 $\forall m \in M. \forall m'. \text{justified } m' m \longrightarrow m' \in M$
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* :
 $\forall n m m'. n \in \mathbb{N}$
 $\longrightarrow \text{justified } m' m$
 $\longrightarrow m \in \text{Mi } (V, C, \varepsilon) n$
 $\longrightarrow m' \in \text{Mi } (V, C, \varepsilon) (n - 1)$
proof –
have $\forall n m m'. \text{justified } m' m$
 $\longrightarrow m \in \text{Mi } (V, C, \varepsilon) n$
 $\longrightarrow m \in M \wedge m' \in M$
 $\longrightarrow m' \in \text{Mi } (V, C, \varepsilon) (n - 1)$
apply (*rule, rule, rule, rule, rule, rule*)
proof –
fix $n m m'$
assume *justified m' m*
assume $m \in \text{Mi } (V, C, \varepsilon) n$
assume $m \in M \wedge m' \in M$
then have *justification m* $\in \Sigma i (V, C, \varepsilon) n$
using *Mi.simps* $\langle m \in \text{Mi } (V, C, \varepsilon) n \rangle$ **by** *blast*
then have *justification m* $\in \text{Pow } (\text{Mi } (V, C, \varepsilon) (n - 1))$
by (*metis (no-types, lifting) Suc-diff-Suc* *$\Sigma i.simps(1)$* *$\Sigma i\text{-subset-Mi}$* $\langle \text{justified } m' m \rangle$ *add-leE* *diff-add* *diff-le-self* *empty-iff justified-def* *neq0-conv* *plus-1-eq-Suc singletonD* *subsetCE*)
show $m' \in \text{Mi } (V, C, \varepsilon) (n - 1)$
using $\langle \text{justification } m \in \text{Pow } (\text{Mi } (V, C, \varepsilon) (n - 1)) \rangle \langle \text{justified } m' m \rangle$
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* :
 $\forall m m'. m \in M$
 $\longrightarrow \text{justified } m' m$
 $\longrightarrow \text{card } (\text{justification } m') < \text{card } (\text{justification } m)$
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 $\neg \text{wfp-on justified } M$
then have $\exists f. \forall i. f i \in M \wedge \text{justified } (f (\text{Suc } i)) (f i)$
by (*simp add: wfp-on-def*)
then obtain *f* **where** $\forall i. f i \in M \wedge \text{justified } (f (\text{Suc } i)) (f i)$ **by** *auto*
have $\forall i. \text{card } (\text{justification } (f i)) \leq \text{card } (\text{justification } (f 0)) - i$
apply (*rule*)
proof –
fix *i*
have $\text{card } (\text{justification } (f (\text{Suc } i))) < \text{card } (\text{justification } (f i))$
using $\langle \forall i. f i \in M \wedge \text{justified } (f (\text{Suc } i)) (f i) \rangle$ **by** (*simp add: monotonicity-of-card-of-justification*)
show $\text{card } (\text{justification } (f i)) \leq \text{card } (\text{justification } (f 0)) - i$
apply (*induction i*)
apply *simp*
using $\langle \text{card } (\text{justification } (f (\text{Suc } i))) < \text{card } (\text{justification } (f i)) \rangle$
by (*smt Suc-diff-le $\forall i. f i \in M \wedge \text{justified } (f (\text{Suc } i)) (f i)$ 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 $\exists i. i = \text{card } (\text{justification } (f 0)) + \text{Suc } 0 \wedge \text{card } (\text{justification } (f i)) \leq \text{card } (\text{justification } (f 0)) - i$
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 $\forall i. f i \in M \wedge \text{justified } (f (\text{Suc } i)) (f i)$ add.right-neutral add-Suc-right*)
qed

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

end

3 Latest Message

theory *LatestMessage*

imports *Main CBCCaspar MessageJustification Libraries/LaTeXsugar*

begin

definition *later* :: (message * message set) \Rightarrow message set

where

$later = (\lambda(m, \sigma). \{m' \in \sigma. justified\ m\ m'\})$

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

$from-sender = (\lambda(v, \sigma). \{m \in \sigma. sender\ m = v\})$

lemma (**in** *Protocol*) *from-sender-type* :

$\forall \sigma\ v. \sigma \in Pow\ M \wedge v \in V \longrightarrow from-sender\ (v, \sigma) \in Pow\ M$

apply (*simp add: from-sender-def*)

by *auto*

lemma (**in** *Protocol*) *from-sender-type-for-state* :

$\forall \sigma\ v. \sigma \in \Sigma \wedge v \in V \longrightarrow from-sender\ (v, \sigma) \subseteq M$

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

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

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

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

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

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

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

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

definition *observed-non-equivocating-validators* :: *state* \Rightarrow *validator set*
where
 $\text{observed-non-equivocating-validators } \sigma = \text{observed } \sigma - \text{equivocating-validators } \sigma$

lemma (in *Protocol*) *observed-non-equivocating-validators-type* :
 $\forall \sigma \in \Sigma. \text{observed-non-equivocating-validators } \sigma \in \text{Pow } V$
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*:
 $\forall \sigma \in \Sigma. (\forall v \in V. \text{wfp-on justified (from-sender (v, } \sigma))})$
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*:
 $\forall \sigma \in \Sigma. (\forall v \in V. v \notin \text{equivocating-validators } \sigma \longrightarrow \text{Relation.total-on (from-sender (v, } \sigma)) \text{ message-justification})$
proof –
have $\forall m1 m2 \sigma v. v \in V \wedge \sigma \in \Sigma \wedge \{m1, m2\} \subseteq \text{from-sender (v, } \sigma) \longrightarrow \text{sender } m1 = \text{sender } m2$
by (*simp add: from-sender-def*)
then have $\forall \sigma \in \Sigma. (\forall v \in V. v \notin \text{equivocating-validators } \sigma \longrightarrow (\forall m1 m2. \{m1, m2\} \subseteq \text{from-sender (v, } \sigma) \longrightarrow m1 = m2 \vee \text{justified } m1 m2 \vee \text{justified } m2 m1))$
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*:
 $\forall \sigma \in \Sigma. (\forall v \in V. v \notin \text{equivocating-validators } \sigma \longrightarrow \text{strict-linear-order-on (from-sender (v, } \sigma)) \text{ message-justification})$
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*:
 $\forall \sigma \in \Sigma. (\forall v \in V. v \notin \text{equivocating-validators } \sigma \longrightarrow \text{strict-linear-order-on (from-sender (v, } \sigma)) \text{ message-justification} \wedge \text{wfp-on$

```

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 :
  ∀ σ v. σ ∈ Σ ∧ v ∈ V ⟶ latest-messages σ 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:
  ∀ σ ∈ Σ. (∀ v ∈ observed σ. is-singleton (latest-messages
σ v))
  apply (simp add: observed-non-equivocating-validators-def)
  proof -
    have ∀ σ ∈ Σ. (∀ v ∈ 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 ∀ σ ∈ Σ. ∀ v ∈ observed σ. equivocating-validators σ. is-singleton
(latest-messages σ 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 ∈ latest-messages σ v}

lemma (in *Protocol*) *latest-estimates-type* :
 $\forall \sigma v. \sigma \in \Sigma \wedge v \in V \longrightarrow \text{latest-estimates } \sigma v \subseteq C$
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* :
 $\forall \sigma v. \sigma \in \Sigma \wedge v \in V \wedge v \notin \text{observed } \sigma \longrightarrow \text{latest-estimates } \sigma v = \emptyset$
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
 $\text{latest-messages-from-non-equivocating-validators } \sigma v = (\text{if is-equivocating } \sigma v \text{ then } \emptyset \text{ else latest-messages } \sigma v)$

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

definition *latest-estimates-from-non-equivocating-validators* :: *state* \Rightarrow *validator*
 \Rightarrow *consensus-value set*
where
 $\text{latest-estimates-from-non-equivocating-validators } \sigma v = \{\text{est } m \mid m. m \in \text{latest-messages-from-non-equivocating-validators } \sigma v\}$

lemma (in *Protocol*) *latest-estimates-from-non-equivocating-validators-type* :
 $\forall \sigma v. \sigma \in \Sigma \wedge v \in V \longrightarrow \text{latest-estimates-from-non-equivocating-validators } \sigma v \in \text{Pow } C$
using *Protocol.latest-estimates-type Protocol-axioms latest-estimates-def latest-estimates-from-non-equivocating-validators-def*
by *auto*

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

```

end
theory StateTransition

imports Main CBC_Casper MessageJustification

begin

definition (in Params) state-transition :: state rel
where
  state-transition =  $\{(\sigma 1, \sigma 2). \{\sigma 1, \sigma 2\} \subseteq \Sigma \wedge \text{is-future-state}(\sigma 1, \sigma 2)\}$ 

lemma (in Params) reflexivity-of-state-transition :
  refl-on  $\Sigma$  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  $\Sigma$  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  $\Sigma$  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  $\equiv \{(\sigma, \sigma') \mid \sigma \sigma'. \sigma \in \Sigma t \wedge \sigma' \in \Sigma t \wedge \text{is-future-state}(\sigma, \sigma') \wedge \sigma \neq \sigma' \wedge (\nexists \sigma''. \sigma'' \in \Sigma \wedge \text{is-future-state}(\sigma, \sigma'') \wedge \text{is-future-state}(\sigma'', \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma')\}$ 

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

```

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

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

proof –

fix $n \sigma m$
assume $1 \leq n \sigma \in \Sigma i (V, C, \varepsilon) n m \in Mi (V, C, \varepsilon) n \text{immediately-next-message } (\sigma, m)$

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

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

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

have $\{m\} \in \text{Pow } (Mi (V, C, \varepsilon) n)$
using $\langle m \in Mi (V, C, \varepsilon) n \rangle$ **by** *auto*
moreover **have** $\sigma \in \text{Pow } (Mi (V, C, \varepsilon) (n-1))$
using $\langle \sigma \in \Sigma i (V, C, \varepsilon) n \rangle si$ **by** *auto*
hence $\sigma \in \text{Pow } (Mi (V, C, \varepsilon) n)$
using *Mi-monotonic*
by (*metis (full-types) PowD PowI Suc-eq-plus1 $\langle \exists n'. n = \text{Suc } n' \rangle \text{diff-Suc-1 subset-iff}$*)
ultimately **have** $\sigma \cup \{m\} \in \text{Pow } (Mi (V, C, \varepsilon) n)$
by *blast*

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

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

$\forall \sigma \in \Sigma i (V, C, \varepsilon) n. \forall m \in Mi (V, C, \varepsilon) n. \text{immediately-next-message } (\sigma, m) \longrightarrow \sigma$
 $\cup \{m\} \in \Sigma i (V, C, \varepsilon) (n+1)$
apply (cases n)
apply auto[1]
using *state-transition-by-immediately-next-message-of-same-depth-non-zero*
by (metis le-add1 plus-1-eq-Suc)

lemma (in *Params*) *past-state-exists-in-same-depth* :

$\forall \sigma \sigma'. \sigma' \in \Sigma i (V, C, \varepsilon) n \longrightarrow \sigma \subseteq \sigma' \longrightarrow \sigma \in \Sigma \longrightarrow \sigma \in \Sigma i (V, C, \varepsilon) n$
apply (rule, rule, rule, rule, rule)
proof (cases n)
case 0
show $\bigwedge \sigma \sigma'. \sigma' \in \Sigma i (V, C, \varepsilon) n \implies \sigma \subseteq \sigma' \implies \sigma \in \Sigma \implies n = 0 \implies \sigma \in \Sigma i (V, C, \varepsilon) n$
by auto
next
case (Suc nat)
show $\bigwedge \sigma \sigma' \text{ nat}. \sigma' \in \Sigma i (V, C, \varepsilon) n \implies \sigma \subseteq \sigma' \implies \sigma \in \Sigma \implies n = \text{Suc nat} \implies \sigma \in \Sigma i (V, C, \varepsilon) n$
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 $\text{finite } \sigma \wedge (\forall m. m \in \sigma \longrightarrow \text{justification } m \subseteq \sigma)$
using $\langle \sigma \in \Sigma \rangle$ *state-is-finite state-is-in-pow-Mi* **by** blast
moreover have $\sigma \in \text{Pow } (Mi (V, C, \varepsilon) (n - 1))$
using $\langle \sigma \subseteq \sigma' \rangle$
by (smt Pow-iff Suc-eq-plus1 Σi -monotonic Σi -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 $\sigma \in \{\sigma \in \text{Pow } (Mi (V, C, \varepsilon) (n - 1)). \text{finite } \sigma \wedge (\forall m. m \in \sigma \longrightarrow \text{justification } m \subseteq \sigma)\}$
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*:

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

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

lemma (in *Protocol*) *state-transition-imps-immediately-next-message*:
 $\forall \sigma \in \Sigma. \forall m \in M. \sigma \cup \{m\} \in \Sigma \wedge m \notin \sigma \longrightarrow \text{immediately-next-message } (\sigma, m)$
proof –
have $\forall \sigma \in \Sigma. \forall m \in M. \sigma \cup \{m\} \in \Sigma \longrightarrow (\forall m' \in \sigma \cup \{m\}. \text{justification } m' \subseteq \sigma \cup \{m\})$
using *state-is-in-pow-Mi* **by** *blast*
then have $\forall \sigma \in \Sigma. \forall m \in M. \sigma \cup \{m\} \in \Sigma \longrightarrow \text{justification } m \subseteq \sigma \cup \{m\}$
by *auto*
then have $\forall \sigma \in \Sigma. \forall m \in M. \sigma \cup \{m\} \in \Sigma \wedge m \notin \sigma \longrightarrow \text{justification } m \subseteq \sigma$
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*:
 $\forall \sigma \in \Sigma. \forall m \in M. \sigma \cup \{m\} \in \Sigma \wedge m \notin \sigma \longleftrightarrow \text{immediately-next-message } (\sigma, m)$
using *state-transition-imps-immediately-next-message state-transition-by-immediately-next-message*
apply (*simp add: immediately-next-message-def*)
by *blast*

lemma (in *Protocol*) *state-transition-is-immediately-next-message*:
 $\forall \sigma \in \Sigma. \forall m \in M. \sigma \cup \{m\} \in \Sigma \longleftrightarrow \text{justification } m \subseteq \sigma$
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*:
 $\forall \sigma \in \Sigma. \forall \sigma'. \sigma' \subset \sigma \longrightarrow (\exists m \in \sigma - \sigma'. \text{immediately-next-message } (\sigma', m))$
apply (*simp add: immediately-next-message-def*)

```

    apply (rule, rule, rule)
  proof -
    fix  $\sigma \sigma'$ 
    assume  $\sigma \in \Sigma$ 
    assume  $\sigma' \subset \sigma$ 
    show  $\exists m \in \sigma - \sigma'. \text{justification } m \subseteq \sigma'$ 
    proof (rule ccontr)
      assume  $\neg (\exists m \in \sigma - \sigma'. \text{justification } m \subseteq \sigma')$ 
      then have  $\forall m \in \sigma - \sigma'. \exists m' \in \text{justification } m. m' \in \sigma - \sigma'$ 
        using  $\langle \neg (\exists m \in \sigma - \sigma'. \text{justification } m \subseteq \sigma') \rangle \text{state-is-in-pow-Mi } \langle \sigma' \subset \sigma \rangle$ 
        by (metis Diff-iff  $\langle \sigma \in \Sigma \rangle$  subset-eq)
      then have  $\forall m \in \sigma - \sigma'. \exists m'. \text{justified } m' m \wedge m' \in \sigma - \sigma'$ 
        using justified-def by auto
      then have  $\forall m \in \sigma - \sigma'. \exists m'. \text{justified } m' m \wedge m' \in \sigma - \sigma' \wedge m \neq m'$ 
        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  $\langle \sigma \in \Sigma \rangle \langle \sigma' \subset \sigma \rangle \text{state-is-subset-of-M}$  by auto
      then have  $\exists m\text{-min} \in \sigma - \sigma'. \forall m. \text{justified } m m\text{-min} \longrightarrow m \notin \sigma - \sigma'$ 
        using subset-of-M-have-minimal-of-justification  $\langle \sigma' \subset \sigma \rangle$ 
        by blast
      then show False
        using  $\langle \forall m \in \sigma - \sigma'. \exists m'. \text{justified } m' m \wedge m' \in \sigma - \sigma' \rangle$  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  $\forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \longrightarrow (\exists m \in \sigma - (\sigma \cap \sigma'). \text{immediately-next-message}(\sigma \cap \sigma', m))$ 
            by (metis Int-subset-iff psubsetI strict-subset-of-state-have-immediately-next-messages subsetI)
          then have  $\forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \longrightarrow (\exists m \in \sigma - (\sigma \cap \sigma'). \text{immediately-next-message}(\sigma', m))$ 
            immediately-next-message( $\sigma', m$ )
          apply (simp add: immediately-next-message-def)
          by blast
    qed
  
```


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 $\forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \longrightarrow \text{card } (\sigma - \sigma') > 0$
by (*meson Diff-eq-empty-iff card-0-eq finite-Diff gr0I state-is-finite*)
have $\forall n. \forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \wedge \text{Suc } n = \text{card } (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
apply (*rule*)
proof –
fix n
show $\forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \wedge \text{Suc } n = \text{card } (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
apply (*induction n*)
apply (*rule, rule, rule*)
proof –
fix $\sigma \sigma'$
assume $\sigma \in \Sigma$ **and** $\sigma' \in \Sigma$ **and** $\neg \sigma \subseteq \sigma' \wedge \text{Suc } 0 = \text{card } (\sigma - \sigma')$
then have *is-singleton* $(\sigma - \sigma')$
by (*simp add: is-singleton-altdef*)
then have $\{ \text{the-elem } (\sigma - \sigma') \} \cup \sigma' \in \Sigma$
using $\langle \forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \longrightarrow (\exists m \in \sigma - \sigma'. \sigma' \cup \{m\} \in \Sigma) \rangle \langle \sigma \in \Sigma \rangle \langle \sigma' \in \Sigma \rangle$
by (*metis Un-commute* $\langle \neg \sigma \subseteq \sigma' \wedge \text{Suc } 0 = \text{card } (\sigma - \sigma') \rangle$ *is-singleton-the-elem singletonD*)
then show $\sigma \cup \sigma' \in \Sigma$
by (*metis Un-Diff-cancel2* $\langle \text{is-singleton } (\sigma - \sigma') \rangle$ *is-singleton-the-elem*)

next
show $\bigwedge n. \forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \wedge \text{Suc } n = \text{card } (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma \implies \forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \wedge \text{Suc } (\text{Suc } n) = \text{card } (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
apply (*rule, rule, rule*)
proof –
fix $n \sigma \sigma'$
assume $\forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \wedge \text{Suc } n = \text{card } (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$ **and** $\sigma \in \Sigma$ **and** $\sigma' \in \Sigma$ **and** $\neg \sigma \subseteq \sigma' \wedge \text{Suc } (\text{Suc } n) = \text{card } (\sigma - \sigma')$
have $\forall m \in \sigma - \sigma'. \neg \sigma \subseteq \sigma' \cup \{m\} \wedge \text{Suc } n = \text{card } (\sigma - (\sigma' \cup \{m\}))$
using $\langle \neg \sigma \subseteq \sigma' \wedge \text{Suc } (\text{Suc } n) = \text{card } (\sigma - \sigma') \rangle$
by (*metis Diff-eq-empty-iff Diff-insert Un-insert-right* $\langle \sigma \in \Sigma \rangle$ *add-diff-cancel-left'* *card-0-eq card-Suc-Diff1 finite-Diff nat.simps(3)* *plus-1-eq-Suc state-is-finite sup-bot.right-neutral*)
have $\exists m \in \sigma - \sigma'. \sigma' \cup \{m\} \in \Sigma$
using $\langle \forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \longrightarrow (\exists m \in \sigma - \sigma'. \sigma' \cup \{m\} \in \Sigma) \rangle \langle \sigma \in \Sigma \rangle \langle \sigma' \in \Sigma \rangle \langle \neg \sigma \subseteq \sigma' \wedge \text{Suc } (\text{Suc } n) = \text{card } (\sigma - \sigma') \rangle$
by *blast*
then have $\exists m \in \sigma - \sigma'. \sigma' \cup \{m\} \in \Sigma \wedge \neg \sigma \subseteq \sigma' \cup \{m\} \wedge \text{Suc } n = \text{card } (\sigma - (\sigma' \cup \{m\}))$
using $\langle \forall m \in \sigma - \sigma'. \neg \sigma \subseteq \sigma' \cup \{m\} \wedge \text{Suc } n = \text{card } (\sigma - (\sigma' \cup \{m\})) \rangle$

```

      by simp
    then show  $\sigma \cup \sigma' \in \Sigma$ 
      using  $\langle \forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \wedge \text{Suc } n = \text{card } (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma \rangle$ 
      by (smt Un-Diff-cancel Un-commute Un-insert-right  $\langle \sigma \in \Sigma \rangle$ 
insert-absorb2 mk-disjoint-insert sup-bot.right-neutral)
    qed
  qed
  then show ?thesis
    by (meson  $\langle \forall \sigma \in \Sigma. \forall \sigma' \in \Sigma. \neg \sigma \subseteq \sigma' \longrightarrow (\exists m \in \sigma - \sigma'. \sigma' \cup \{m\} \in \Sigma) \rangle$ 
card-Suc-Diff1 finite-Diff state-is-finite)
  qed
  then show ?thesis
    using False  $\langle \sigma 1 \in \Sigma \rangle \langle \sigma 2 \in \Sigma \rangle$  by blast
  qed
qed

```

lemma (in Protocol) *union-of-finite-set-of-states-is-state* :

```

   $\forall \sigma\text{-set} \subseteq \Sigma. \text{finite } \sigma\text{-set} \longrightarrow \bigcup \sigma\text{-set} \in \Sigma$ 
  apply auto
  proof -
    have  $\forall n. \forall \sigma\text{-set} \subseteq \Sigma. n = \text{card } \sigma\text{-set} \longrightarrow \text{finite } \sigma\text{-set} \longrightarrow \bigcup \sigma\text{-set} \in \Sigma$ 
      apply (rule)
    proof -
      fix n
      show  $\forall \sigma\text{-set} \subseteq \Sigma. n = \text{card } \sigma\text{-set} \longrightarrow \text{finite } \sigma\text{-set} \longrightarrow \bigcup \sigma\text{-set} \in \Sigma$ 
        apply (induction n)
        apply (rule, rule, rule, rule)
        apply (simp add: empty-set-exists-in- $\Sigma$ )
        apply (rule, rule, rule, rule)
      proof -
        fix n  $\sigma\text{-set}$ 
        assume  $\forall \sigma\text{-set} \subseteq \Sigma. n = \text{card } \sigma\text{-set} \longrightarrow \text{finite } \sigma\text{-set} \longrightarrow \bigcup \sigma\text{-set} \in \Sigma$  and
 $\sigma\text{-set} \subseteq \Sigma$  and  $\text{Suc } n = \text{card } \sigma\text{-set}$  and  $\text{finite } \sigma\text{-set}$ 
        then have  $\forall \sigma \in \sigma\text{-set}. \sigma\text{-set} - \{\sigma\} \subseteq \Sigma \wedge \bigcup (\sigma\text{-set} - \{\sigma\}) \in \Sigma$ 
          using  $\langle \sigma\text{-set} \subseteq \Sigma \rangle \langle \text{Suc } n = \text{card } \sigma\text{-set} \rangle \langle \forall \sigma\text{-set} \subseteq \Sigma. n = \text{card } \sigma\text{-set} \longrightarrow \text{finite } \sigma\text{-set} \longrightarrow \bigcup \sigma\text{-set} \in \Sigma \rangle$ 
          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  $\langle \sigma\text{-set} \subseteq \Sigma \rangle$  by auto
        then show  $\bigcup \sigma\text{-set} \in \Sigma$ 
          by (metis Sup-bot-conv(1) Sup-insert Un-commute empty-set-exists-in- $\Sigma$ 
insert-Diff)
      qed
    qed
  qed

```

then show $\bigwedge \sigma\text{-set}. \sigma\text{-set} \subseteq \Sigma \implies \text{finite } \sigma\text{-set} \implies \bigcup \sigma\text{-set} \in \Sigma$
by *blast*
qed

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

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

lemma (in *Protocol*) *minimal-transition-implies-recieving-single-message* :
 $\forall \sigma\ \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \longrightarrow \text{is-singleton } (\sigma' - \sigma)$
proof (*rule ccontr*)
assume $\neg (\forall \sigma\ \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \longrightarrow \text{is-singleton } (\sigma' - \sigma))$
then have $\exists \sigma\ \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma)$
by *blast*
have $\forall \sigma\ \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \longrightarrow$
 $(\nexists \sigma''. \sigma'' \in \Sigma \wedge \text{is-future-state } (\sigma, \sigma'') \wedge \text{is-future-state } (\sigma'', \sigma') \wedge \sigma$
 $\neq \sigma'' \wedge \sigma'' \neq \sigma')$
by (*simp add: minimal-transitions-def*)
have $\forall \sigma\ \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma)$
 $\longrightarrow (\exists m1\ m2. \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$
 $\text{immediately-next-message } (\sigma, m1))$
apply (*rule, rule, rule*)
proof –
fix $\sigma\ \sigma'$
assume $(\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma)$
then have $\sigma' - \sigma \neq \emptyset$
apply (*simp add: minimal-transitions-def*)
by *blast*
have $\sigma' \in \Sigma \wedge \sigma \in \Sigma \wedge \text{is-future-state } (\sigma, \sigma')$
using $\langle (\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma) \rangle$
by (*simp add: minimal-transitions-def Σt-def*)
then have $\sigma' - \sigma \subseteq M$
using *state-difference-is-valid-message* **by** *auto*
then have $\exists m1\ m2. \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1$
 $\neq m2$
using *non-empty-non-singleton-imps-two-elements*
 $\langle (\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma) \rangle \langle \sigma' - \sigma \neq \emptyset \rangle$
by (*metis (full-types) contra-subsetD insert-subset subsetI*)
then show $\exists m1\ m2. \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1$
 $\neq m2 \wedge \text{immediately-next-message } (\sigma, m1)$
using *state-differences-have-immediately-next-messages*

by (*metis Diff-iff* $\langle \sigma' \in \Sigma \wedge \sigma \in \Sigma \wedge \text{is-future-state } (\sigma, \sigma') \rangle \text{ insert-subset message-in-state-is-valid}$)
qed
have $\forall \sigma \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma) \longrightarrow$
 $(\exists \sigma''. \sigma'' \in \Sigma \wedge \text{is-future-state } (\sigma, \sigma'') \wedge \text{is-future-state } (\sigma'', \sigma') \wedge \sigma$
 $\neq \sigma'' \wedge \sigma'' \neq \sigma')$
apply (*rule, rule, rule*)
proof –
fix $\sigma \sigma'$
assume $(\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma)$
then have $\exists m1 m2. \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq$
 $m2 \wedge \text{immediately-next-message } (\sigma, m1)$
using $\langle \forall \sigma \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma)$
 $\longrightarrow (\exists m1 m2. \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$
 $\text{immediately-next-message } (\sigma, m1)) \rangle$
by simp
then obtain $m1 m2$ **where** $\{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge$
 $m1 \neq m2 \wedge \text{immediately-next-message } (\sigma, m1)$
by auto
have $\sigma \in \Sigma \wedge \sigma' \in \Sigma$
using $\langle (\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma) \rangle$
by (*simp add: minimal-transitions-def Σt -def*)
then have $\sigma \cup \{m1\} \in \Sigma$
using $\langle \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$
 $\text{immediately-next-message } (\sigma, m1) \rangle$
state-transition-by-immediately-next-message
by simp
have $\text{is-future-state } (\sigma, \sigma \cup \{m1\}) \wedge \text{is-future-state } (\sigma \cup \{m1\}, \sigma')$
using $\langle (\sigma, \sigma') \in \text{minimal-transitions} \wedge \neg \text{is-singleton } (\sigma' - \sigma) \rangle \langle \{m1, m2\} \subseteq$
 $M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge \text{immediately-next-message } (\sigma,$
 $m1) \rangle \text{ minimal-transitions-def}$ **by auto**
have $\sigma \neq \sigma \cup \{m1\} \wedge \sigma \cup \{m1\} \neq \sigma'$
using $\langle \{m1, m2\} \subseteq M \wedge m1 \in \sigma' - \sigma \wedge m2 \in \sigma' - \sigma \wedge m1 \neq m2 \wedge$
 $\text{immediately-next-message } (\sigma, m1) \rangle$ **by auto**
then show $\exists \sigma''. \sigma'' \in \Sigma \wedge \text{is-future-state } (\sigma, \sigma'') \wedge \text{is-future-state } (\sigma'', \sigma') \wedge$
 $\sigma \neq \sigma'' \wedge \sigma'' \neq \sigma'$
using $\langle \sigma \cup \{m1\} \in \Sigma \rangle \langle \text{is-future-state } (\sigma, \sigma \cup \{m1\}) \wedge \text{is-future-state } (\sigma \cup$
 $\{m1\}, \sigma') \rangle$
by auto
qed
then show *False*
using $\langle \forall \sigma \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \longrightarrow (\nexists \sigma''. \sigma'' \in \Sigma \wedge \text{is-future-state}$
 $(\sigma, \sigma'') \wedge \text{is-future-state } (\sigma'', \sigma') \wedge \sigma \neq \sigma'' \wedge \sigma'' \neq \sigma') \rangle \langle \neg (\forall \sigma \sigma'. (\sigma, \sigma') \in$
 $\text{minimal-transitions} \longrightarrow \text{is-singleton } (\sigma' - \sigma)) \rangle$ **by blast**
qed

lemma (*in Protocol*) *minimal-transitions-reconstruction* :

$\forall \sigma \sigma'. (\sigma, \sigma') \in \text{minimal-transitions} \longrightarrow \sigma \cup \{\text{the-elem } (\sigma' - \sigma)\} = \sigma'$
apply (*rule, rule, rule*)

```

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

end

```

4 Safety Proof

theory *ConsensusSafety*

imports *Main CBCCasper MessageJustification StateTransition Libraries/LaTeXsugar*

begin

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

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

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

theorem $(\text{in } \text{Protocol}) \text{ n-party-common-futures} :$
 $\forall \ \sigma\text{-set}. \ \sigma\text{-set} \subseteq \Sigma t$
 $\longrightarrow \text{finite } \sigma\text{-set}$
 $\longrightarrow \text{is-faults-lt-threshold } (\bigcup \sigma\text{-set})$

$\longrightarrow \bigcap \{ \text{futures } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \} \neq \emptyset$
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 :
 $\forall \sigma\text{-set}. \sigma\text{-set} \subseteq \Sigma t$
 $\longrightarrow \text{finite } \sigma\text{-set}$
 $\longrightarrow \text{is-faults-lt-threshold } (\bigcup \sigma\text{-set})$
 $\longrightarrow (\exists \sigma \in \Sigma t. \sigma \in \bigcap \{ \text{futures } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \})$
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
 $\text{state-property-is-decided} = (\lambda(p, \sigma). (\forall \sigma' \in \text{futures } \sigma. p \sigma'))$

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

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

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

theorem (in Protocol) two-party-consensus-safety-for-state-property :
 $\forall \sigma 1 \sigma 2. \sigma 1 \in \Sigma t \wedge \sigma 2 \in \Sigma t$
 $\longrightarrow \text{is-faults-lt-threshold } (\sigma 1 \cup \sigma 2)$
 $\longrightarrow \neg (\text{state-property-is-decided } (p, \sigma 1) \wedge \text{state-property-is-decided } (\text{state-property-not } p, \sigma 2))$
apply (simp add: state-property-is-decided-def)

using *two-party-common-futures*
by (*metis Int-emptyI*)

definition (**in** *Protocol*) *state-properties-are-inconsistent* :: *state-property set* \Rightarrow *bool*
where
state-properties-are-inconsistent p-set = $(\forall \sigma \in \Sigma. \neg (\forall p \in p\text{-set}. p \sigma))$

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

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

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

proof–
fix $\sigma\text{-set}$
assume $\sigma\text{-set} : \sigma\text{-set} \subseteq \Sigma t$
and *finite* $\sigma\text{-set}$
and *is-faults-lt-threshold* $(\bigcup \sigma\text{-set})$
hence $\exists \sigma \in \Sigma t. \sigma \in \bigcap \{\text{futures } \sigma \mid \sigma. \sigma \in \sigma\text{-set}\}$
using *n-party-common-futures-exists* **by** *simp*
hence $\exists \sigma \in \Sigma t. \forall s \in \sigma\text{-set}. \sigma \in \text{futures } s$
by *blast*
hence $\exists \sigma \in \Sigma t. (\forall s \in \sigma\text{-set}. \sigma \in \text{futures } s) \wedge (\forall s \in \sigma\text{-set}. \sigma \in \text{futures } s \longrightarrow (\forall p. \text{state-property-is-decided } (p, s) \longrightarrow \text{state-property-is-decided } (p, \sigma)))$
by (*simp add: subset-eq state-property-is-decided-def futures-def*)
hence $\exists \sigma \in \Sigma t. \forall s \in \sigma\text{-set}. (\forall p. \text{state-property-is-decided } (p, s) \longrightarrow \text{state-property-is-decided } (p, \sigma))$
by *blast*
hence $\exists \sigma \in \Sigma t. \forall s \in \sigma\text{-set}. (\forall p \in \text{state-property-decisions } s. \text{state-property-is-decided } (p, \sigma))$
by (*simp add: state-property-decisions-def*)
hence $\exists \sigma \in \Sigma t. \forall p \in \bigcup \{\text{state-property-decisions } \sigma \mid \sigma. \sigma \in \sigma\text{-set}\}. \text{state-property-is-decided } (p, \sigma)$
proof–
obtain σ **where** $\sigma \in \Sigma t \wedge \forall s \in \sigma\text{-set}. (\forall p \in \text{state-property-decisions } s. \text{state-property-is-decided } (p, \sigma))$

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

definition (in Protocol) *naturally-corresponding-state-property* :: *consensus-value-property* \Rightarrow *state-property*

where

naturally-corresponding-state-property $q = (\lambda \sigma. \forall c \in \varepsilon \sigma. q \ c)$

definition (in Protocol) *consensus-value-properties-are-consistent* :: *consensus-value-property* *set* \Rightarrow *bool*

where

consensus-value-properties-are-consistent $q\text{-set} = (\exists c \in C. \forall q \in q\text{-set}. q \ c)$

lemma (in Protocol) *naturally-corresponding-consistency* :

$\forall q\text{-set}. \text{state-properties-are-consistent } \{ \text{naturally-corresponding-state-property } q \mid q. q \in q\text{-set} \}$

\longrightarrow *consensus-value-properties-are-consistent* $q\text{-set}$

apply (*rule, rule*)

proof –

fix $q\text{-set}$

have

$\text{state-properties-are-consistent } \{ \text{naturally-corresponding-state-property } q \mid q. q \in q\text{-set} \}$

$\longrightarrow (\exists \sigma \in \Sigma. \forall p \in \{ \lambda \sigma'. \forall c \in \varepsilon \sigma'. q \ c \mid q. q \in q\text{-set} \}. p \ \sigma)$

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
 $state\text{-properties-are-consistent} \ \{naturally\text{-corresponding-state-property} \ q \mid q. \ q \in q\text{-set}\}$
 $\implies consensus\text{-value-properties-are-consistent} \ q\text{-set}$
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
 $consensus\text{-value-property-is-decided}$
 $= (\lambda(q, \sigma). state\text{-property-is-decided} \ (naturally\text{-corresponding-state-property} \ q, \sigma))$

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

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

```

and finite  $\sigma$ -set
and is-faults-lt-threshold ( $\bigcup \sigma$ -set)
hence state-properties-are-consistent ( $\bigcup \{ \text{state-property-decisions } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \}$ )
using  $\langle \sigma\text{-set} \subseteq \Sigma t \rangle$  n-party-safety-for-state-properties by auto
hence state-properties-are-consistent  $\{ p \in \bigcup \{ \text{state-property-decisions } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \}. \exists q. p = \text{naturally-corresponding-state-property } q \}$ 
unfolding naturally-corresponding-state-property-def state-properties-are-consistent-def
apply (simp)
by meson
hence state-properties-are-consistent  $\{ \text{naturally-corresponding-state-property } q \mid q. \text{naturally-corresponding-state-property } q \in \bigcup \{ \text{state-property-decisions } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \} \}$ 
by (smt Collect-cong)
hence consensus-value-properties-are-consistent  $\{ q. \text{naturally-corresponding-state-property } q \in \bigcup \{ \text{state-property-decisions } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \} \}$ 
using naturally-corresponding-consistency
proof –
show ?thesis
by (metis (no-types) Setcompr-eq-image  $\langle \forall q\text{-set}. \text{state-properties-are-consistent } \{ \text{naturally-corresponding-state-property } q \mid q. q \in q\text{-set} \} \longrightarrow \text{consensus-value-properties-are-consistent } q\text{-set} \rangle$   $\langle \text{state-properties-are-consistent } \{ \text{naturally-corresponding-state-property } q \mid q. \text{naturally-corresponding-state-property } q \in \bigcup \{ \text{state-property-decisions } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \} \} \rangle$  setcompr-eq-image)
qed
hence consensus-value-properties-are-consistent ( $\bigcup \{ \text{consensus-value-property-decisions } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \}$ )
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 ( $\bigcup \{ \text{consensus-value-property-decisions } \sigma \mid \sigma. \sigma \in \sigma\text{-set} \}$ )
by simp
qed

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

lemma (in Protocol) negation-is-not-decided-by-other-validator :
 $\forall \sigma\text{-set}. \sigma\text{-set} \subseteq \Sigma t$ 
 $\longrightarrow$  finite  $\sigma\text{-set}$ 
 $\longrightarrow$  is-faults-lt-threshold ( $\bigcup \sigma\text{-set}$ )
 $\longrightarrow (\forall \sigma \sigma' p. \{ \sigma, \sigma' \} \subseteq \sigma\text{-set} \wedge p \in \text{consensus-value-property-decisions } \sigma \longrightarrow \text{consensus-value-property-not } p \notin \text{consensus-value-property-decisions } \sigma')$ 
apply (rule, rule, rule, rule, rule, rule, rule, rule)
proof –

```

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

using *backward-consistency* $\langle \text{state-property-is-decided } (\text{naturally-corresponding-state-property } p, \sigma'') \rangle$
using $\langle \sigma'' \in \Sigma t \wedge \sigma'' \in \bigcap \text{-Collect } (\text{futures } \sigma) (\sigma \in \sigma\text{-set}) \rangle$ $\langle \sigma\text{-set} \subseteq \Sigma t \rangle$ $\langle \{\sigma, \sigma'\} \subseteq \sigma\text{-set} \wedge p \in \text{consensus-value-property-decisions } \sigma \rangle$ **by** *auto*
then show *consensus-value-property-not* $p \notin \text{consensus-value-property-decisions } \sigma'$
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 $\Sigma t\text{-def}$ *estimates-are-non-empty futures-def* **by** *fastforce*
qed

lemma (in *Protocol*) *n-party-consensus-safety* :

$\forall \sigma\text{-set}. \sigma\text{-set} \subseteq \Sigma t$
 \rightarrow *finite* $\sigma\text{-set}$
 \rightarrow *is-faults-lt-threshold* $(\bigcup \sigma\text{-set})$
 $\rightarrow (\forall p \in \bigcup \{\text{consensus-value-property-decisions } \sigma' \mid \sigma'. \sigma' \in \sigma\text{-set}\}.$
 $(\lambda c. (\neg p \ c)) \notin \bigcup \{\text{consensus-value-property-decisions } \sigma' \mid \sigma'. \sigma' \in \sigma\text{-set}\})$
apply (*rule, rule, rule, rule, rule, rule*)

proof –

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

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

lemma (in *Protocol*) *two-party-consensus-safety-for-consensus-value-property* :

$\forall \sigma 1 \ \sigma 2. \sigma 1 \in \Sigma t \wedge \sigma 2 \in \Sigma t$
 $\longrightarrow \text{is-faults-lt-threshold } (\sigma 1 \cup \sigma 2)$
 $\longrightarrow \text{consensus-value-property-is-decided } (p, \sigma 1)$
 $\longrightarrow \neg \text{consensus-value-property-is-decided } (\text{consensus-value-property-not } p, \sigma 2)$
apply $(\text{rule, rule, rule, rule, rule})$

proof –

fix $\sigma 1 \ \sigma 2$
have *two-party*: $\forall \sigma 1 \ \sigma 2. \{\sigma 1, \sigma 2\} \subseteq \Sigma t$
 $\longrightarrow \text{is-faults-lt-threshold } (\bigcup \{\sigma 1, \sigma 2\})$
 $\longrightarrow p \in \text{consensus-value-property-decisions } \sigma 1$
 $\longrightarrow \text{consensus-value-property-not } p \notin \text{consensus-value-property-decisions}$

$\sigma 2$

using *negation-is-not-decided-by-other-validator*

by $(\text{meson finite.emptyI finite.insertI order-refl})$

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 $\neg \text{consensus-value-property-is-decided } (\text{consensus-value-property-not } p, \sigma 2)$

using *two-party*

apply $(\text{simp add: consensus-value-property-decisions-def})$

by *blast*

qed

lemma (in *Protocol*) *n-party-consensus-safety-for-power-set-of-decisions* :

$\forall \sigma\text{-set}. \sigma\text{-set} \subseteq \Sigma t$
 $\longrightarrow \text{finite } \sigma\text{-set}$
 $\longrightarrow \text{is-faults-lt-threshold } (\bigcup \sigma\text{-set})$
 $\longrightarrow (\forall \sigma \ p\text{-set}. \sigma \in \sigma\text{-set} \wedge p\text{-set} \in \text{Pow } (\bigcup \{ \text{consensus-value-property-decisions } \sigma' \mid \sigma'. \sigma' \in \sigma\text{-set} \}) - \{\emptyset\})$
 $\longrightarrow (\lambda c. \neg (\forall p \in p\text{-set}. p \ c)) \notin \text{consensus-value-property-decisions } \sigma$
apply $(\text{rule, rule, rule, rule, rule, rule, rule, rule})$

proof –

fix $\sigma\text{-set } \sigma \ p\text{-set}$

assume $\sigma\text{-set} \subseteq \Sigma t$ **and** *finite* $\sigma\text{-set}$ **and** *is-faults-lt-threshold* $(\bigcup \sigma\text{-set})$

```

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

end
theory SafetyOracle

imports Main CBCCaspar LatestMessage StateTransition

begin

```

fun *latest-justifications-from-non-equivocating-validators* :: *state* \Rightarrow *validator* \Rightarrow *state set*
where
latest-justifications-from-non-equivocating-validators σ v =
 $\{ \text{justification } m \mid m. m \in \text{latest-messages-from-non-equivocating-validators } \sigma \ v \}$

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

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

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

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

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

definition (**in** *Params*) *weight-measure* :: *validator set* \Rightarrow *real*
where
weight-measure $v\text{-set}$ = *sum* $W \ v\text{-set}$

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

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

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

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

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

fun *is-clique* :: (*validator set* * *consensus-value-property* * *state*) \Rightarrow *bool*
where
is-clique $(v\text{-set}, p, \sigma) =$
 $(\forall v \in v\text{-set}. v\text{-set} \subseteq \text{agreeing-validators} \ (p, \text{the-elem} \ (\text{latest-justifications-from-non-equivocating-validators} \ \sigma \ v))$
 $\wedge (\forall v' \in v\text{-set}. \text{later-disagreeing-messages} \ (p, \text{the-elem} \ (\text{latest-messages-from-non-equivocating-validators} \ (\text{the-elem} \ (\text{latest-justifications-from-non-equivocating-validators} \ \sigma \ v)) \ v'), v', \sigma) = \emptyset))$

lemma (in *Protocol*) *later-from-not-affected-by-minimal-transitions* :
 $\forall \sigma \ \sigma' \ m \ m' \ v. (\sigma, \sigma') \in \text{minimal-transitions}$
 $\longrightarrow m' = \text{the-elem} \ (\sigma' - \sigma)$
 $\longrightarrow v \in V - \{\text{sender } m'\}$
 $\longrightarrow \text{later-from} \ (m, v, \sigma) = \text{later-from} \ (m, v, \sigma')$
apply (*rule*, *rule*, *rule*, *rule*, *rule*, *rule*, *rule*)

```

proof–
  fix  $\sigma \ \sigma' \ m \ m' \ v$ 
  assume  $(\sigma, \sigma') \in \text{minimal-transitions}$ 
  assume  $m' = \text{the-elem } (\sigma' - \sigma)$ 
  assume  $v \in V - \{\text{sender } m'\}$ 

  have  $\text{later-from } (m, v, \sigma) = \{m'' \in \sigma. \text{ sender } m'' = v \wedge \text{justified } m \ m''\}$ 
  apply (simp add: later-from-def from-sender-def later-def)
  by auto
  also have  $\dots = \{m'' \in \sigma. \text{ sender } m'' = v \wedge \text{justified } m \ m''\} \cup \emptyset$ 
  by auto
  also have  $\dots = \{m'' \in \sigma. \text{ sender } m'' = v \wedge \text{justified } m \ m''\} \cup \{m'' \in \{m'\}. \text{ sender } m'' = v\}$ 
proof–
  have  $\{m'' \in \{m'\}. \text{ sender } m'' = v\} = \emptyset$ 
  using  $\langle v \in V - \{\text{sender } m'\} \rangle$  by auto
  thus ?thesis
  by blast
qed
  also have  $\dots = \{m'' \in \sigma. \text{ sender } m'' = v \wedge \text{justified } m \ m''\} \cup \{m'' \in \{m'\}. \text{ sender } m'' = v \wedge \text{justified } m \ m''\}$ 
proof–
  have  $\text{sender } m' = v \implies \text{justified } m \ m'$ 
  using  $\langle v \in V - \{\text{sender } m'\} \rangle$  by auto
  thus ?thesis
  by blast
qed
  also have  $\dots = \{m'' \in \sigma \cup \{m'\}. \text{ sender } m'' = v \wedge \text{justified } m \ m''\}$ 
  by auto
  also have  $\dots = \{m'' \in \sigma'. \text{ sender } m'' = v \wedge \text{justified } m \ m''\}$ 
proof–
  have  $\sigma' = \sigma \cup \{m'\}$ 
  using  $\langle (\sigma, \sigma') \in \text{minimal-transitions} \rangle \langle m' = \text{the-elem } (\sigma' - \sigma) \rangle$  minimal-transitions-reconstruction
by auto
  then show ?thesis
  by auto
qed
  then have  $\dots = \text{later-from } (m, v, \sigma')$ 
  apply (simp add: later-from-def from-sender-def later-def)
  by auto
  then show  $\text{later-from } (m, v, \sigma) = \text{later-from } (m, v, \sigma')$ 
  using  $\langle \{m'' \in \sigma \cup \{m'\}. \text{ sender } m'' = v \wedge \text{justified } m \ m''\} = \{m'' \in \sigma'. \text{ sender } m'' = v \wedge \text{justified } m \ m''\} \rangle$  calculation by auto
qed

fun (in Params) gt-threshold :: (validator set * state)  $\Rightarrow$  bool
  where
    gt-threshold (v-set,  $\sigma$ )

```


$= (\text{weight-measure } v\text{-set} > (\text{weight-measure } v\text{-set}) \text{ div } 2 + t - \text{weight-measure } (\text{equivocating-validators } \sigma))$

```

fun (in Params) is-clique-oracle :: (validator set * state * consensus-value-property)
⇒ bool
  where
    is-clique-oracle (v-set,  $\sigma$ , p)
      = (is-clique (v-set − (equivocating-validators  $\sigma$ ), p,  $\sigma$ ) ∧ gt-threshold (v-set
      − (equivocating-validators  $\sigma$ ),  $\sigma$ ))

end
theory TFGCasper

```

```

imports Main HOL.Real CBCCasper LatestMessage SafetyOracle ConsensusSafety

```

```

begin

```

```

type-synonym block = consensus-value

```

```

locale GhostParams = Params +

```

```

  fixes B :: block set
  fixes genesis :: block

```

```

  and prev :: block ⇒ block

```

```

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

```

```

definition (in GhostParams) 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 GhostParams) score :: state ⇒ block ⇒ real
  where
    score  $\sigma$  b =  $\text{sum } W \{v \in \text{observed } \sigma. \exists b' \in B. b' \in (\text{latest-estimates-from-non-equivocating-validators } \sigma \ v) \wedge (b \mid b')\}$ 

```

definition (in *GhostParams*) *children* :: *block* * *state* \Rightarrow *block set*
where
children = $(\lambda(b, \sigma). \{b' \in \text{est} \mid \sigma. b = \text{prev } b'\})$

definition (in *GhostParams*) *best-children* :: *block* * *state* \Rightarrow *block set*
where
best-children = $(\lambda(b, \sigma). \{\arg\text{-max-on } (\text{score } \sigma) (\text{children } (b, \sigma))\})$

function (in *GhostParams*) *GHOST* :: (*block set* * *state*) \Rightarrow *block set*
where
GHOST (*b-set*, σ) =
 $(\bigcup b \in \{b \in \text{b-set}. \text{children } (b, \sigma) \neq \emptyset\}. \text{GHOST } (\text{best-children } (b, \sigma), \sigma))$
 $\cup \{b \in \text{b-set}. \text{children } (b, \sigma) = \emptyset\}$
by *auto*

definition (in *GhostParams*) *GHOST-estimator* :: *state* \Rightarrow *block set*
where
GHOST-estimator $\sigma = \text{GHOST } (\{\text{genesis}\}, \sigma) \cup (\bigcup b \in \text{GHOST } (\{\text{genesis}\}, \sigma). \text{children } (b, \sigma))$

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

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

definition (in *GhostParams*) *block-membership-property* :: *block* \Rightarrow *consensus-value-property*
where
block-membership-property $b = (\lambda b'. b \downarrow b')$

definition (in *GhostParams*) *block-conflicting* :: (*block* * *block*) \Rightarrow *bool*
where
block-conflicting = $(\lambda(b1, b2). \neg (b1 \downarrow b2 \vee b2 \downarrow b1))$

lemma (in *Blockchain*) *conflicting-blocks-implies-conflicting-decision* :
 $\forall b1 \ b2 \ \sigma. \{b1, b2\} \subseteq B \wedge \sigma \in \Sigma \longrightarrow \text{block-conflicting } (b1, b2)$

\rightarrow consensus-value-property-is-decided (block-membership-property b1, σ)
 \rightarrow 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** $\forall \sigma \in \text{futures}$
 $\sigma. \forall b' \in \varepsilon \sigma. b1 \downarrow b'$
show $\forall \sigma \in \text{futures} \sigma. \forall c \in \varepsilon \sigma. \neg b2 \downarrow c$
proof (rule ccontr)
assume $\neg (\forall \sigma \in \text{futures} \sigma. \forall c \in \varepsilon \sigma. \neg b2 \downarrow c)$
hence $\exists \sigma \in \text{futures} \sigma. \exists c \in \varepsilon \sigma. b2 \downarrow c$
by blast
hence $\exists \sigma \in \text{futures} \sigma. \exists c \in \varepsilon \sigma. b2 \downarrow c \wedge b1 \downarrow c$
using $\langle \forall \sigma \in \text{futures} \sigma. \forall b' \in \varepsilon \sigma. b1 \downarrow b' \rangle$ **by** simp
hence $b1 \downarrow b2 \vee b2 \downarrow b1$
using blockchain-type
apply (simp)
using Σt -is-subset-of- Σ $\langle b1 \in B \wedge b2 \in B \wedge \sigma \in \Sigma \rangle$ block-is-consensus-value
estimates-are-subset-of-C futures-def **by** blast
then show False
using $\langle \text{block-conflicting} (b1, b2) \rangle$
by (simp add: block-conflicting-def)
qed
qed

theorem (in Blockchain) blockchain-safety :

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

using $\langle \{\sigma, \sigma'\} \subseteq \sigma\text{-set} \wedge \{b1, b2\} \subseteq B \wedge \text{block-conflicting} (b1, b2) \wedge$
 $\text{block-membership-property } b1 \in \text{consensus-value-property-decisions } \sigma \rangle$ **by** auto

have $\{b1, b2\} \subseteq B \wedge \sigma \in \Sigma \wedge \text{block-conflicting } (b1, b2)$
using $\Sigma t\text{-is-subset-of-}\Sigma \langle \sigma\text{-set} \subseteq \Sigma t \rangle \langle \{\sigma, \sigma'\} \subseteq \sigma\text{-set} \wedge \{b1, b2\} \subseteq B \wedge$
 $\text{block-conflicting } (b1, b2) \wedge \text{block-membership-property } b1 \in \text{consensus-value-property-decisions}$
 $\sigma \rangle$ **by** *auto*
hence $\text{consensus-value-property-is-decided } (\text{consensus-value-property-not } (\text{block-membership-property}$
 $b1), \sigma')$
using $\langle \text{block-membership-property } b2 \in \text{consensus-value-property-decisions } \sigma' \rangle$
 $\text{conflicting-blocks-implies-conflicting-decision}$
apply (*simp add: consensus-value-property-decisions-def*)
by (*metis* $\langle \sigma\text{-set} \subseteq \Sigma t \rangle \langle \text{finite } \sigma\text{-set} \rangle \langle \text{is-faults-lt-threshold } (\bigcup \sigma\text{-set}) \rangle \langle \{\sigma, \sigma'\} \subseteq$
 $\sigma\text{-set} \wedge \{b1, b2\} \subseteq B \wedge \text{block-conflicting } (b1, b2) \wedge \text{block-membership-property } b1$
 $\in \text{consensus-value-property-decisions } \sigma \rangle \text{conflicting-blocks-implies-conflicting-decision}$
 $\text{consensus-value-property-decisions-def insert-subset mem-Collect-eq negation-is-not-decided-by-other-validator}$)

then show *False*
using $\langle \neg \text{consensus-value-property-is-decided } (\text{consensus-value-property-not}$
 $(\text{block-membership-property } b1), \sigma') \rangle$ **by** *blast*
qed

theorem (*in Blockchain*) *no-decision-on-conflicting-blocks* :

$\forall \sigma 1 \sigma 2. \{ \sigma 1, \sigma 2 \} \subseteq \Sigma t$
 $\longrightarrow \text{is-faults-lt-threshold } (\sigma 1 \cup \sigma 2)$
 $\longrightarrow (\forall b1 b2. \{ b1, b2 \} \subseteq C \wedge \text{block-conflicting } (b1, b2)$
 $\longrightarrow \text{block-membership-property } b1 \in \text{consensus-value-property-decisions } \sigma 1$
 $\longrightarrow \text{block-membership-property } b2 \notin \text{consensus-value-property-decisions } \sigma 2)$
apply (*rule, rule, rule, rule, rule, rule, rule, rule, rule*)
proof –
fix $\sigma 1 \sigma 2 b1 b2$
assume $\{ \sigma 1, \sigma 2 \} \subseteq \Sigma t$ **and** $\text{is-faults-lt-threshold } (\sigma 1 \cup \sigma 2)$ **and** $\{ b1, b2 \} \subseteq C$
 $\wedge \text{block-conflicting } (b1, b2)$
and $\text{block-membership-property } b1 \in \text{consensus-value-property-decisions } \sigma 1$
and $\text{block-membership-property } b2 \in \text{consensus-value-property-decisions } \sigma 2$
hence $\text{consensus-value-property-is-decided } (\text{block-membership-property } b1, \sigma 1)$
by (*simp add: consensus-value-property-decisions-def*)
hence $\neg \text{consensus-value-property-is-decided } (\text{consensus-value-property-not } (\text{block-membership-property}$
 $b1), \sigma 2)$
using *two-party-consensus-safety-for-consensus-value-property* $\langle \text{is-faults-lt-threshold}$
 $(\sigma 1 \cup \sigma 2) \rangle \langle \{ \sigma 1, \sigma 2 \} \subseteq \Sigma t \rangle$ **by** *blast*
have $\text{block-membership-property } b2 \in \text{consensus-value-property-decisions } \sigma 2$
using $\langle \text{block-membership-property } b2 \in \text{consensus-value-property-decisions } \sigma 2 \rangle$

by (*simp add: consensus-value-property-decisions-def*)
have $\sigma 2 \in \Sigma t \wedge \{ b2, b1 \} \subseteq B \wedge \text{block-conflicting } (b2, b1)$
using $\text{block-is-consensus-value } \langle \{ \sigma 1, \sigma 2 \} \subseteq \Sigma t \rangle \langle \{ b1, b2 \} \subseteq C \wedge \text{block-conflicting}$
 $(b1, b2) \rangle$ **by** (*simp add: block-conflicting-def*)
hence $\text{consensus-value-property-is-decided } (\text{consensus-value-property-not } (\text{block-membership-property}$
 $b1), \sigma 2)$
using $\text{conflicting-blocks-implies-conflicting-decision } \langle \text{block-membership-property}$

```

b2  $\in$  consensus-value-property-decisions  $\sigma 2$ 
  using  $\Sigma$ -is-subset-of- $\Sigma$  consensus-value-property-decisions-def by auto
  then show False
    using  $\langle \neg$  consensus-value-property-is-decided (consensus-value-property-not
      (block-membership-property b1),  $\sigma 2$ )  $\rangle$  by blast
  qed

```

```

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

```

```

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

```

```

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

```

```

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

```

```

lemma (in GhostParams) GHOST-is-valid-estimator :
   $(\forall b. b \in B \longleftrightarrow \text{prev } b \in B) \wedge B = C \wedge \text{genesis} \in C$ 
   $\implies \text{is-valid-estimator GHOST-estimator}$ 
  apply (simp add: is-valid-estimator-def GhostParams.GHOST-estimator-def)
  oops

```

```

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

  oops

```

```

lemma (in Ghost) block-membership-property-is-max-driven :

```

```

 $\forall p \in P. \text{is-max-driven } p$ 
apply (simp add: is-max-driven-def)

oops

end
```