# Minimal CBC Casper Isabelle/HOL proofs

LayerX

February 27, 2019

## Contents

**1 Description of CBC Casper** 3

**2 Latest Message** 14

**3 Safety Proof** 29

**theory** *Strict-Order*

**imports** *Main*

**begin**

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

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

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

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

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

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

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

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

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

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

**by** (*metis* (*no-types*, *lifting*) *Un-insert-right* ‹∀ *a*. (*a*, *a*) ∉ *r*› ‹*strict-partial-order r*› *insertE insert-iff strict-partial-order-def sup-bot.right-neutral transE*)
**qed**
**qed**
**then show** *?thesis*
**by** (*metis card.insert-remove finite.cases*)
**qed**

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

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

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

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

**end**

# 1 Description of CBC Casper

**theory** *CBCCasper*

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

**begin**

**notation** *Set.empty* (∅)

**typedecl** *validator*

**typedecl** *consensus-value*

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

**type-synonym** *state* = *message set*

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

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

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

**fun**
  Σ-*i* :: (*validator set* × *consensus-value set* × (*message set* ⇒ *consensus-value set*)) ⇒ *nat* ⇒ *state set* **and**
  *M-i* :: (*validator set* × *consensus-value set* × (*message set* ⇒ *consensus-value set*)) ⇒ *nat* ⇒ *message set*
  **where**
    Σ-*i* (*V*,*C*,ε) *0* = {∅}
  | Σ-*i* (*V*,*C*,ε) *n* = {σ ∈ *Pow* (*M-i* (*V*,*C*,ε) (*n* − *1*)). *finite* σ ∧ (∀ *m*. *m* ∈ σ ⟶ *justification m* ⊆ σ)}
  | *M-i* (*V*,*C*,ε) *n* = {*m*. *est m* ∈ *C* ∧ *sender m* ∈ *V* ∧ *justification m* ∈ (Σ-*i*

$(V,C,\varepsilon)\ n) \wedge \mathit{est}\ m \in \varepsilon\ (\mathit{justification}\ m)\}$

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

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

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

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

  **lemma** *Mi-subset-to-*$\Sigma$*i*: $M\text{-}i\ (V,C,\varepsilon)\ n \subseteq M\text{-}i\ (V,C,\varepsilon)\ (n+1) \Longrightarrow \Sigma\text{-}i\ (V,C,\varepsilon)\ (n+1) \subseteq \Sigma\text{-}i\ (V,C,\varepsilon)\ (n+2)$
    **by** *auto*

  **lemma** $\Sigma$*i-monotonic*: $\Sigma\text{-}i\ (V,C,\varepsilon)\ n \subseteq \Sigma\text{-}i\ (V,C,\varepsilon)\ (n+1)$
    **apply** (*induction n*)
    **apply** *simp*
   **apply** (*metis Mi-subset-to-*$\Sigma$*i Suc-eq-plus1* $\Sigma$*i-subset-to-Mi add.commute add-2-eq-Suc*)
    **done**

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

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

  **lemma** *Mi-monotonicity*: $\forall\ m \in \mathbb{N}.\ \forall\ n \in \mathbb{N}.\ m \leq n \longrightarrow M\text{-}i\ (V,C,\varepsilon)\ m \subseteq M\text{-}i\ (V,C,\varepsilon)\ n$
    **using** *Mi-monotonic*

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

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

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

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

**lemma** *message-in-state-is-valid* :
$\forall\ \sigma\ m.\ \sigma \in \Sigma \wedge m \in \sigma \longrightarrow\ m \in M$
**apply** (*rule*, *rule*, *rule*)
**proof** −
**fix** $\sigma$ *m*
**assume** $\sigma \in \Sigma \wedge m \in \sigma$
**have**
$\exists\ n \in \mathbb{N}.\ m \in M\text{-}i\ (V,\ C,\ \varepsilon)\ n$
$\Longrightarrow m \in M$
**using** *M-def* **by** *blast*
**then show**
$m \in M$

**apply** (*simp add*: *M-def*)
**by** (*smt M-i.simps Params.*$\Sigma$*i-monotonic PowD Suc-diff-Suc* ‹$\sigma \in \Sigma \wedge m \in \sigma$› *add-leE diff-add diff-le-self gr0I mem-Collect-eq plus-1-eq-Suc state-is-in-pow-M-i subsetCE zero-less-diff*)
**qed**

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

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

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

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

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

**end**

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

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

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

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

**lemma** (**in** *Params*) *empty-set-exists-in-*$\Sigma$: $\emptyset \in \Sigma$
**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*:
$\exists\ m.\ m \in M\text{-}i\ (V,\ C,\ \varepsilon)\ 0 \land justification\ m = \emptyset$
**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** $\exists m.\ est\ m \in C \land sender\ m \in V \land justification\ m = \emptyset \land est\ m \in \varepsilon$ $(justification\ m) \land justification\ m = \emptyset$
**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*) *M-i-is-non-empty*: $M\text{-}i\ (V,\ C,\ \varepsilon)\ n \neq \emptyset$
**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*) *M-is-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-*$\Sigma$ :
$\forall\ n \in \mathbb{N}.\ \Sigma\text{-}i\ (V,\ C,\ \varepsilon)\ n \subseteq \Sigma$
**by** (*simp add*: $\Sigma$*-def SUP-upper*)

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

**lemma** (**in** *Protocol*) $\Sigma$-*type*: $\Sigma \subset$ *Pow M*
**proof** $-$
**obtain** $m$ **where** $m \in M$-*i* $(V, C, \varepsilon)$ $0 \wedge$ *justification* $m = \emptyset$
**using** *non-justifying-message-exists-in-M-0* **by** *auto*
**then have** $\{m\} \in \Sigma$-*i* $(V, C, \varepsilon)$ (*Suc 0*)
**using** *Params.*$\Sigma$*i-subset-Mi* **by** *auto*
**then have** $\exists$ $m'$. $m' \in$ *M-i* $(V, C, \varepsilon)$ (*Suc 0*) $\wedge$ *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** $m' \in$ *M-i* $(V, C, \varepsilon)$ (*Suc 0*) $\wedge$ *justification* $m' = \{m\}$ **by** *auto*
**then have** $\{m'\} \in$ *Pow M*
**using** *M-def*
**by** (*metis Nats-1 One-nat-def PowD PowI Pow-bottom UN-I insert-subset*)
**moreover have** $\{m'\} \notin \Sigma$
**using** *Params.state-is-in-pow-M-i Protocol-axioms* $\langle m' \in$ *M-i* $(V, C, \varepsilon)$ (*Suc 0*) $\wedge$ *justification* $m' = \{m\}\rangle$ **by** *fastforce*
**ultimately show** *?thesis*
**using** $\Sigma$*-is-subseteq-of-pow-M* **by** *auto*
**qed**

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

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

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

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

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

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

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

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

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

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

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

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

**lemma** (**in** *Protocol*) *equivocating-validators-is-equivalent-to-paper* :

$\forall\ \sigma \in \Sigma.$ *equivocating-validators* $\sigma$ = *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* ⇒ *real*
**where**
*equivocation-fault-weight* $\sigma$ = *sum W* (*equivocating-validators* $\sigma$)

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

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

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

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

**type-synonym** *consensus-value-property* = *consensus-value* ⇒ *bool*

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

**lemma** (**in** *Protocol*) *transitivity-of-justifications* :
*trans message-justification*
**apply** (*simp add*: *trans-def message-justification-def justified-def*)
**by** (*meson Params.M-type Params.state-is-in-pow-M-i 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* $\in$ *M-i* (*V*, *C*, $\varepsilon$) (*n* $-$ *1*)
**by** (*smt M-i.simps One-nat-def Params.*$\Sigma$*i-subset-Mi Pow-iff Suc-pred* ⟨*est m* $\in$ *C*⟩ ⟨*est m* $\in$ $\varepsilon$ (*justification m*)⟩ ⟨*justification m* $\in$ $\Sigma$-*i* (*V*, *C*, $\varepsilon$) *n*⟩ ⟨*m* $\in$ *justification m*⟩ ⟨*sender m* $\in$ *V*⟩ *add.right-neutral add-Suc-right diff-is-0-eq′ diff-le-self diff-zero mem-Collect-eq not-gr0 subsetCE*)
**then have** *justification m* $\in$ $\Sigma$-*i* (*V*, *C*, $\varepsilon$) (*n* $-$ *1*)
**using** *M-i.simps* **by** *blast*
**then have** *justification m* $\in$ $\Sigma$-*i* (*V*, *C*, $\varepsilon$) *0*
**apply** (*induction n*)
**apply** *simp*
**by** (*smt M-i.simps One-nat-def Params.*$\Sigma$*i-subset-Mi Pow-iff Suc-pred* ⟨*m* $\in$ *justification m*⟩ *add.right-neutral add-Suc-right diff-Suc-1 mem-Collect-eq not-gr0 subsetCE subsetCE*)
**then have** *justification m* $\in$ $\{\emptyset\}$
**by** *simp*
**then show** *False*
**using** ⟨*m* $\in$ *justification m*⟩ **by** *blast*
**qed**

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

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

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

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

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

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

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

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

**lemma** (**in** *Protocol*) *justification-is-well-founded-on-M* :

*wfp-on justified M*
**proof** (*rule ccontr*)
**assume** ¬ *wfp-on justified M*
**then have** $\exists f.\ \forall i.\ f\ i \in M \wedge \mathit{justified}\ (f\ (\mathit{Suc}\ i))\ (f\ i)$
**by** (*simp add*: *wfp-on-def*)
**then obtain** *f* **where** $\forall i.\ f\ i \in M \wedge \mathit{justified}\ (f\ (\mathit{Suc}\ i))\ (f\ i)$ **by** *auto*
**have** $\forall\ i.\ \mathit{card}\ (\mathit{justification}\ (f\ i)) \leq \mathit{card}\ (\mathit{justification}\ (f\ 0)) - i$
**apply** (*rule*)
**proof** −
**fix** *i*
**have** $\mathit{card}\ (\mathit{justification}\ (f\ (\mathit{Suc}\ i))) < \mathit{card}\ (\mathit{justification}\ (f\ i))$
**using** ‹$\forall i.\ f\ i \in M \wedge \mathit{justified}\ (f\ (\mathit{Suc}\ i))\ (f\ i)$› **by** (*simp add*: *monotonicity-of-card-of-justification*)
**show** $\mathit{card}\ (\mathit{justification}\ (f\ i)) \leq \mathit{card}\ (\mathit{justification}\ (f\ 0)) - i$
**apply** (*induction i*)
**apply** *simp*
**using** ‹$\mathit{card}\ (\mathit{justification}\ (f\ (\mathit{Suc}\ i))) < \mathit{card}\ (\mathit{justification}\ (f\ i))$›
**by** (*smt Suc-diff-le* ‹$\forall i.\ f\ i \in M \wedge \mathit{justified}\ (f\ (\mathit{Suc}\ i))\ (f\ i)$› *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** $\exists\ i.\ i = \mathit{card}\ (\mathit{justification}\ (f\ 0)) + \mathit{Suc}\ 0 \wedge \mathit{card}\ (\mathit{justification}\ (f\ i)) \leq \mathit{card}\ (\mathit{justification}\ (f\ 0)) - i$
**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* ‹$\forall i.\ f\ i \in M \wedge \mathit{justified}\ (f\ (\mathit{Suc}\ i))\ (f\ i)$› *add.right-neutral add-Suc-right*)
**qed**

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

**end**

# 2 Latest Message

**theory** *LatestMessage*

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

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

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

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

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

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

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

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

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

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

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

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

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

**lemma** (**in** *Protocol*) *justification-is-well-founded-on-messages-from-validator*:
$\forall\ \sigma \in \Sigma.\ (\forall\ v \in V.\ \ \textit{wfp-on justified}\ (\textit{from-sender}\ (v, \sigma)))$
**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*:
$\forall\ \sigma \in \Sigma.\ (\forall\ v \in V.\ v \notin \textit{equivocating-validators}\ \sigma \longrightarrow \textit{Relation.total-on}\ (\textit{from-sender}\ (v, \sigma))\ \textit{message-justification})$
**proof** −
**have** $\forall\ m1\ m2\ \sigma\ v.\ v \in V \wedge \sigma \in \Sigma \wedge \{m1, m2\} \subseteq \textit{from-sender}\ (v, \sigma) \longrightarrow \textit{sender}\ m1 = \textit{sender}\ m2$
**by** (*simp add*: *from-sender-def*)
**then have** $\forall\ \sigma \in \Sigma.\ (\forall\ v \in V.\ v \notin \textit{equivocating-validators}\ \sigma \longrightarrow (\forall\ m1\ m2.\ \{m1, m2\} \subseteq \textit{from-sender}\ (v, \sigma) \longrightarrow m1 = m2 \vee \textit{justified}\ m1\ m2 \vee \textit{justified}\ m2\ m1))$
**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*:
$\forall\ \sigma \in \Sigma.\ (\forall\ v \in V.\ v \notin \textit{equivocating-validators}\ \sigma \longrightarrow \textit{strict-linear-order-on}\ (\textit{from-sender}\ (v, \sigma))\ \textit{message-justification})$
**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*:
$\forall\ \sigma \in \Sigma.\ (\forall\ v \in V.\ v \notin \textit{equivocating-validators}\ \sigma \longrightarrow \textit{strict-linear-order-on}\ (\textit{from-sender}\ (v, \sigma))\ \textit{message-justification} \wedge \textit{wfp-on justified}\ (\textit{from-sender}\ (v, \sigma)))$
**using** *justification-is-well-founded-on-messages-from-validator*
*justification-is-strict-linear-order-on-messages-from-non-equivocating-validator*
**by** *blast*

**lemma** (**in** *Protocol*) *latest-message-is-maximal-element-of-justification* :
$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow \textit{latest-messages}\ \sigma\ v = \{m.\ \textit{maximal-on}\ (\textit{from-sender}\ (v, \sigma))\ \textit{message-justification}\ m\}$
**apply** (*simp add*: *latest-messages-def later-from-def later-def message-justification-def maximal-on-def*)
**using** *from-sender-type-for-state* **apply** *auto*
**apply** (*metis* (*no-types*, *lifting*) *IntI empty-iff from-sender-def mem-Collect-eq prod.simps*(*2*))

**by** *blast*

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

**by** *fastforce*
**qed**

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

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

**lemma** (**in** *Protocol*) *latest-messages-from-non-equivocating-validators-type* :
  $\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *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**
    *latest-estimates-from-non-equivocating-validators* $\sigma$ $v$ = $\{$*est* $m$ $|$ $m$. $m \in$ *latest-messages-from-non-equivocating-validators* $\sigma$ $v\}$

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

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

**end**
**theory** *StateTransition*

**imports** *Main CBCCasper*

**begin**

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

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

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

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

$\forall n{\geq}1.\ \forall \sigma{\in}\Sigma$-*i* $(V,C,\varepsilon)$ $n$. $\forall m{\in}M$-*i* $(V,C,\varepsilon)$ $n$. *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* $(V, C, \varepsilon)$ $n$ $m \in M$-*i* $(V, C, \varepsilon)$ $n$ *immediately-next-message* $(\sigma, m)$

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

**by** *force*

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

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

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

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

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

$\forall \sigma{\in}\Sigma\text{-}i\ (V,C,\varepsilon)\ n.\ \forall m{\in}M\text{-}i\ (V,C,\varepsilon)\ n.\ immediately\text{-}next\text{-}message\ (\sigma,m) \longrightarrow \sigma \cup \{m\} \in \Sigma\text{-}i\ (V,C,\varepsilon)\ (n+1)$
**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* :
$\forall\ \sigma\ \sigma'.\ \sigma' \in \Sigma\text{-}i\ (V,C,\varepsilon)\ n \longrightarrow \sigma \subseteq \sigma' \longrightarrow \sigma \in \Sigma \longrightarrow \sigma \in \Sigma\text{-}i\ (V,C,\varepsilon)\ n$
**apply** (*rule*, *rule*, *rule*, *rule*, *rule*)
**proof** (*cases n*)
**case** *0*

**show** $\bigwedge\sigma\ \sigma'.\ \sigma' \in \Sigma\text{-}i\ (V,\ C,\ \varepsilon)\ n \Longrightarrow \sigma \subseteq \sigma' \Longrightarrow \sigma \in \Sigma \Longrightarrow n = 0 \Longrightarrow \sigma \in \Sigma\text{-}i\ (V,\ C,\ \varepsilon)\ n$
**by** *auto*
**next**
**case** (*Suc nat*)
**show** $\bigwedge\sigma\ \sigma'\ nat.\ \sigma' \in \Sigma\text{-}i\ (V,\ C,\ \varepsilon)\ n \Longrightarrow \sigma \subseteq \sigma' \Longrightarrow \sigma \in \Sigma \Longrightarrow n = Suc\ nat \Longrightarrow \sigma \in \Sigma\text{-}i\ (V,\ C,\ \varepsilon)\ n$
**proof** −
**fix** $\sigma\ \sigma'$
**assume** $\sigma' \in \Sigma\text{-}i\ (V,\ C,\ \varepsilon)\ n$
**and** $\sigma \subseteq \sigma'$
**and** $\sigma \in \Sigma$
**have** $n > 0$
**by** (*simp add*: *Suc*)
**have** *finite* $\sigma \wedge (\forall\ m.\ m \in \sigma \longrightarrow$ *justification* $m \subseteq \sigma)$
**using** ‹$\sigma \in \Sigma$› *state-is-finite state-is-in-pow-M-i* **by** *blast*
**moreover have** $\sigma \in Pow\ (M\text{-}i\ (V,\ C,\ \varepsilon)\ (n - 1))$
**using** ‹$\sigma \subseteq \sigma'$›
**by** (*smt Pow-iff Suc-eq-plus1* $\Sigma$*i-monotonic* $\Sigma$*i-subset-Mi* ‹$\sigma' \in \Sigma\text{-}i\ (V,\ C,\ \varepsilon)\ n$› *add-diff-cancel-left*′ *add-eq-if diff-is-0-eq diff-le-self plus-1-eq-Suc subset-iff*)
**ultimately have** $\sigma \in \{\sigma \in Pow\ (M\text{-}i\ (V,C,\varepsilon)\ (n - 1)).\ finite\ \sigma \wedge (\forall\ m.\ m \in \sigma \longrightarrow justification\ m \subseteq \sigma)\}$
**by** *blast*
**then show** $\sigma \in \Sigma\text{-}i\ (V,\ C,\ \varepsilon)\ n$
**by** (*simp add*: *Suc*)
**qed**
**qed**

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

**lemma** (**in** *Protocol*) *state-transition-by-immediately-next-message*:
$\forall\ \sigma \in \Sigma.\ \forall\ m \in M.$ *immediately-next-message* $(\sigma,m) \longrightarrow \sigma \cup \{m\} \in \Sigma$
**apply** (*rule*, *rule*, *rule*)
**proof** −
**fix** $\sigma\ m$
**assume** $\sigma \in \Sigma$
**and** $m \in M$
**and** *immediately-next-message* $(\sigma,\ m)$
**then have** $(\exists\ n \in \mathbb{N}.\ \sigma \in \Sigma\text{-}i\ (V,C,\varepsilon)\ n \wedge m \in M\text{-}i\ (V,C,\varepsilon)\ n)$
**using** *immediately-next-message-exists-in-same-depth* ‹$\sigma \in \Sigma$› ‹$m \in M$›
**by** *blast*
**then have** $\exists\ n \in \mathbb{N}.\ \sigma \cup \{m\} \in \Sigma\text{-}i\ (V,C,\varepsilon)\ (n + 1)$
**using** *state-transition-by-immediately-next-message-of-same-depth*
**using** ‹*immediately-next-message* $(\sigma,\ m)$› **by** *blast*

**show** $\sigma \cup \{m\} \in \Sigma$
**apply** (*simp add*: $\Sigma$-*def*)
**by** (*metis Nats-1 Nats-add Un-insert-right* ‹$\exists n{\in}\mathbb{N}$. $\sigma \cup \{m\} \in \Sigma$-*i* $(V, C, \varepsilon)$ $(n + 1)$› *sup-bot.right-neutral*)
**qed**

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

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

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

**lemma** (**in** *Protocol*) *strict-subset-of-state-have-immediately-next-messages*:
$\forall\ \sigma \in \Sigma.\ \forall\ \sigma'.\ \sigma' \subset \sigma \longrightarrow (\exists\ m \in \sigma - \sigma'.$ *immediately-next-message* $(\sigma', m))$
**apply** (*simp add*: *immediately-next-message-def*)
**apply** (*rule*, *rule*, *rule*)
**proof** −
**fix** $\sigma\ \sigma'$
**assume** $\sigma \in \Sigma$
**assume** $\sigma' \subset \sigma$
**show** $\exists\ m \in \sigma - \sigma'.$ *justification* $m \subseteq \sigma'$
**proof** (*rule ccontr*)
**assume** $\neg\ (\exists\ m \in \sigma - \sigma'.$ *justification* $m \subseteq \sigma')$
**then have** $\forall\ m \in \sigma - \sigma'.\ \exists\ m' \in$ *justification* $m.\ m' \in \sigma - \sigma'$
**using** ‹$\neg\ (\exists m{\in}\sigma - \sigma'.$ *justification* $m \subseteq \sigma')$› *state-is-in-pow-M-i* ‹$\sigma' \subset \sigma$›
**by** (*metis Diff-iff* ‹$\sigma \in \Sigma$› *subset-eq*)
**then have** $\forall\ m \in \sigma - \sigma'.\ \exists\ m'.$ *justified* $m'\ m \wedge m' \in \sigma - \sigma'$
**using** *justified-def* **by** *auto*
**then have** $\forall\ m \in \sigma - \sigma'.\ \exists\ m'.$ *justified* $m'\ m \wedge m' \in \sigma - \sigma' \wedge m \neq m'$

    **using** *justification-implies-different-messages state-difference-is-valid-message message-in-state-is-valid* ⟨$\sigma' \subset \sigma$⟩
      **by** (*meson DiffD1* ⟨$\sigma \in \Sigma$⟩)
    **have** $\sigma - \sigma' \subseteq M$
      **using** ⟨$\sigma \in \Sigma$⟩ ⟨$\sigma' \subset \sigma$⟩ *state-is-subset-of-M* **by** *auto*
    **then have** $\exists$ *m-min* $\in \sigma - \sigma'$. $\forall$ *m*. *justified m m-min* $\longrightarrow m \notin \sigma - \sigma'$
      **using** *subset-of-M-have-minimal-of-justification* ⟨$\sigma' \subset \sigma$⟩
      **by** *blast*
    **then show** *False*
      **using** ⟨$\forall\ m \in \sigma - \sigma'.\ \exists\ m'.$ *justified* $m'\ m \wedge m' \in \sigma - \sigma'$⟩ **by** *blast*
  **qed**
**qed**

**lemma** (**in** *Protocol*) *union-of-two-states-is-state* :
  $\forall\ \sigma 1 \in \Sigma.\ \forall\ \sigma 2 \in \Sigma.\ (\sigma 1 \cup \sigma 2) \in \Sigma$
  **apply** (*rule*, *rule*)
**proof** −
  **fix** $\sigma 1\ \sigma 2$
  **assume** $\sigma 1 \in \Sigma$ **and** $\sigma 2 \in \Sigma$
  **show** $\sigma 1 \cup \sigma 2 \in \Sigma$
  **proof** (*cases* $\sigma 1 \subseteq \sigma 2$)
    **case** *True*
    **then show** *?thesis*
      **by** (*simp add*: *Un-absorb1* ⟨$\sigma 2 \in \Sigma$⟩)
  **next**
    **case** *False*
    **then have** $\neg\ \sigma 1 \subseteq \sigma 2$ **by** *simp*
    **have** $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - (\sigma \cap \sigma').$ *immediately-next-message*$(\sigma \cap \sigma', m))$
     **by** (*metis Int-subset-iff psubsetI strict-subset-of-state-have-immediately-next-messages subsetI*)
      **then have** $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - (\sigma \cap \sigma').$ *immediately-next-message*$(\sigma', m))$
      **apply** (*simp add*: *immediately-next-message-def*)
      **by** *blast*
    **then have** $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)$
      **using** *state-transition-by-immediately-next-message*
      **by** (*metis DiffD1 DiffD2 DiffI IntI message-in-state-is-valid*)
    **have** $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow\ \sigma \cup \sigma' \in \Sigma$
    **proof** −
      **have** $\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow$ *card* $(\sigma - \sigma') > 0$
        **by** (*meson Diff-eq-empty-iff card-0-eq finite-Diff gr0I state-is-finite*)
      **have** $\forall\ n.\ \forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* $n =$ *card* $(\sigma - \sigma') \longrightarrow\ \sigma \cup \sigma' \in \Sigma$
        **apply** (*rule*)
      **proof** −
        **fix** *n*
        **show** $\forall \sigma \in \Sigma.\ \forall \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge$ *Suc* $n =$ *card* $(\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
          **apply** (*induction n*)

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

**next**
**show** $\bigwedge n.\ \forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge Suc\ n = card\ (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma \Longrightarrow \forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge Suc\ (Suc\ n) = card\ (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$
**apply** (*rule*, *rule*, *rule*)
**proof** −
**fix** $n$ $\sigma$ $\sigma'$
**assume** $\forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge Suc\ n = card\ (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma$ **and** $\sigma \in \Sigma$ **and** $\sigma' \in \Sigma$ **and** $\neg\ \sigma \subseteq \sigma' \wedge Suc\ (Suc\ n) = card\ (\sigma - \sigma')$
**have** $\forall\ m \in \sigma - \sigma'.\ \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge Suc\ n = card\ (\sigma - (\sigma' \cup \{m\}))$
**using** $\langle\neg\ \sigma \subseteq \sigma' \wedge Suc\ (Suc\ n) = card\ (\sigma - \sigma')\rangle$
**by** (*metis Diff-eq-empty-iff Diff-insert Un-insert-right* $\langle\sigma \in \Sigma\rangle$ *add-diff-cancel-left′ card-0-eq card-Suc-Diff1 finite-Diff nat.simps*(*3*) *plus-1-eq-Suc state-is-finite sup-bot.right-neutral*)
**have** $\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma$
**using** $\langle\forall\ \sigma \in \Sigma.\ \forall\ \sigma' \in \Sigma.\ \neg\ \sigma \subseteq \sigma' \longrightarrow (\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma)\rangle$ $\langle\sigma \in \Sigma\rangle$ $\langle\sigma' \in \Sigma\rangle$ $\langle\neg\ \sigma \subseteq \sigma' \wedge Suc\ (Suc\ n) = card\ (\sigma - \sigma')\rangle$
**by** *blast*
**then have** $\exists\ m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma \wedge \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge Suc\ n = card\ (\sigma - (\sigma' \cup \{m\}))$
**using** $\langle\forall\ m \in \sigma - \sigma'.\ \neg\ \sigma \subseteq \sigma' \cup \{m\} \wedge Suc\ n = card\ (\sigma - (\sigma' \cup \{m\}))\rangle$
**by** *simp*
**then show** $\sigma \cup \sigma' \in \Sigma$
**using** $\langle\forall\sigma{\in}\Sigma.\ \forall\sigma'{\in}\Sigma.\ \neg\ \sigma \subseteq \sigma' \wedge Suc\ n = card\ (\sigma - \sigma') \longrightarrow \sigma \cup \sigma' \in \Sigma\rangle$
**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**
**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* ‹$\sigma 1 \in \Sigma$› ‹$\sigma 2 \in \Sigma$› **by** *blast*
**qed**
**qed**

**lemma** (**in** *Protocol*) *union-of-finite-set-of-states-is-state* :
$\forall$ *σ-set* $\subseteq \Sigma$. *finite σ-set* $\longrightarrow \bigcup$ *σ-set* $\in \Sigma$
**apply** *auto*
**proof** −
**have** $\forall$ *n*. $\forall$ *σ-set* $\subseteq \Sigma$. *n* = *card σ-set* $\longrightarrow$ *finite σ-set* $\longrightarrow \bigcup$ *σ-set* $\in \Sigma$
**apply** (*rule*)
**proof** −
**fix** *n*
**show** $\forall$*σ-set*$\subseteq\Sigma$. *n* = *card σ-set* $\longrightarrow$ *finite σ-set* $\longrightarrow \bigcup$*σ-set* $\in \Sigma$
**apply** (*induction n*)
**apply** (*rule*, *rule*, *rule*, *rule*)
**apply** (*simp add*: *empty-set-exists-in-Σ*)
**apply** (*rule*, *rule*, *rule*, *rule*)
**proof** −
**fix** *n σ-set*
**assume** $\forall$*σ-set*$\subseteq\Sigma$. *n* = *card σ-set* $\longrightarrow$ *finite σ-set* $\longrightarrow \bigcup$*σ-set* $\in \Sigma$ **and** *σ-set* $\subseteq \Sigma$ **and** *Suc n* = *card σ-set* **and** *finite σ-set*
**then have** $\forall$ $\sigma \in$ *σ-set*. *σ-set* − $\{\sigma\} \subseteq \Sigma \wedge \bigcup$ (*σ-set* − $\{\sigma\}$) $\in \Sigma$
**using** ‹*σ-set* $\subseteq \Sigma$› ‹*Suc n* = *card σ-set*› ‹$\forall$*σ-set*$\subseteq\Sigma$. *n* = *card σ-set* $\longrightarrow$ *finite σ-set* $\longrightarrow \bigcup$*σ-set* $\in \Sigma$›
**by** (*metis* (*mono-tags*, *lifting*) *Suc-inject card.remove finite-Diff insert-Diff insert-subset*)
**then have** $\forall$ $\sigma \in$ *σ-set*. *σ-set* − $\{\sigma\} \subseteq \Sigma \wedge \bigcup$ (*σ-set* − $\{\sigma\}$) $\in \Sigma \wedge \bigcup$ (*σ-set* − $\{\sigma\}$) $\cup\, \sigma \in \Sigma$
**using** *union-of-two-states-is-state* ‹*σ-set* $\subseteq \Sigma$› **by** *auto*
**then show** $\bigcup$*σ-set* $\in \Sigma$
**by** (*metis Sup-bot-conv*(*1*) *Sup-insert Un-commute empty-set-exists-in-Σ insert-Diff*)
**qed**
**qed**
**then show** $\bigwedge$*σ-set*. *σ-set* $\subseteq \Sigma \Longrightarrow$ *finite σ-set* $\Longrightarrow \bigcup$*σ-set* $\in \Sigma$
**by** *blast*
**qed**

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

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

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

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

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

**end**

# 3 Safety Proof

**theory** *ConsensusSafety*

**imports** *Main CBCCasper StateTransition Libraries/LaTeXsugar*

**begin**

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

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

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

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

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

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

**lemma** (**in** *Protocol*) *forward-consistency* :
$\forall\ \sigma'\ \sigma.\ \sigma' \in \Sigma t \wedge \sigma \in \Sigma t$
$\longrightarrow \sigma' \in$ *futures* $\sigma$
$\longrightarrow$ *state-property-is-decided* $(p, \sigma)$
$\longrightarrow$ *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**
*state-property-not* $p = (\lambda\sigma.\ (\neg\ p\ \sigma))$

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

**theorem** (**in** *Protocol*) *two-party-consensus-safety* :
$\forall\ \sigma 1\ \sigma 2.\ \sigma 1 \in \Sigma t \wedge \sigma 2 \in \Sigma t$
$\longrightarrow$ *is-faults-lt-threshold* $(\sigma 1 \cup \sigma 2)$
$\longrightarrow \neg$(*state-property-is-decided* $(p, \sigma 1) \wedge$ *state-property-is-decided* (*state-property-not* $p$, $\sigma 2$))
**apply** (*simp add*: *state-property-is-decided-def*)
**using** *two-party-common-futures*
**by** (*metis Int-emptyI*)

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

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

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

**theorem** (**in** *Protocol*) *n-party-safety-for-state-properties* :
$\forall$ $\sigma$*-set*. $\sigma$*-set* $\subseteq \Sigma t$
$\longrightarrow$ *finite* $\sigma$*-set*
$\longrightarrow$ *is-faults-lt-threshold* ($\bigcup$ $\sigma$*-set*)
$\longrightarrow$ *state-properties-are-consistent* ($\bigcup$ $\{$*state-property-decisions* $\sigma \mid \sigma. \sigma \in \sigma$*-set*$\}$)
**apply** *rule+*
**proof**−
**fix** $\sigma$*-set*
**assume** $\sigma$*-set*: $\sigma$*-set* $\subseteq \Sigma t$
**and** *finite* $\sigma$*-set*
**and** *is-faults-lt-threshold* ($\bigcup$ $\sigma$*-set*)
**hence** $\exists\sigma\in\Sigma t.$ $\sigma \in \bigcap$ $\{$*futures* $\sigma \mid \sigma. \sigma \in \sigma$*-set*$\}$
**using** *n-party-common-futures-exists*
**by** (*simp add*: ‹*finite* $\sigma$*-set*› ‹*is-faults-lt-threshold* ($\bigcup\sigma$*-set*)› $\sigma$*-set*)
**hence** $\exists\sigma\in\Sigma t.$ $\forall s\in\sigma$*-set*. $\sigma \in$ *futures s*
**by** *blast*
**hence** $\exists\sigma\in\Sigma t.$ ($\forall s\in\sigma$*-set*. $\sigma \in$ *futures s*) $\wedge$ ($\forall s\in\sigma$*-set*. $\sigma \in$ *futures s* $\longrightarrow$ ($\forall p$. *state-property-is-decided* $(p,s)$ $\longrightarrow$ *state-property-is-decided* $(p,\sigma)$))
**by** (*simp add*: *subset-eq state-property-is-decided-def futures-def*)
**hence** $\exists\sigma\in\Sigma t.$ $\forall s\in\sigma$*-set*. ($\forall p$. *state-property-is-decided* $(p,s)$ $\longrightarrow$ *state-property-is-decided* $(p,\sigma)$)
**by** *blast*
**hence** $\exists\sigma\in\Sigma t.$ $\forall s\in\sigma$*-set*. ($\forall p \in$ *state-property-decisions s*. *state-property-is-decided* $(p,\sigma)$)
**by** (*simp add*: *state-property-decisions-def*)
**hence** $\exists\sigma\in\Sigma t.$ $\forall p\in\bigcup\{$*state-property-decisions* $\sigma \mid \sigma. \sigma \in \sigma$*-set*$\}$. *state-property-is-decided* $(p,\sigma)$
**proof**−
**obtain** $\sigma$ **where** $\sigma \in \Sigma t$ $\forall s\in\sigma$*-set*. ($\forall p \in$ *state-property-decisions s*. *state-property-is-decided* $(p,\sigma)$)
**using** ‹$\exists\sigma\in\Sigma t.$ $\forall s\in\sigma$*-set*. $\forall p\in$*state-property-decisions s*. *state-property-is-decided* $(p, \sigma)$› **by** *blast*
**have** $\forall p\in\bigcup\{$*state-property-decisions* $\sigma \mid \sigma. \sigma \in \sigma$*-set*$\}$. *state-property-is-decided* $(p,\sigma)$
**using** ‹$\forall s\in\sigma$*-set*. $\forall p\in$*state-property-decisions s*. *state-property-is-decided* $(p, \sigma)$› **by** *fastforce*
**thus** *?thesis*
**using** ‹$\sigma \in \Sigma t$› **by** *blast*
**qed**
**hence** $\exists\sigma\in\Sigma t.$ $\forall p\in\bigcup\{$*state-property-decisions* $\sigma \mid \sigma. \sigma \in \sigma$*-set*$\}$. $\forall\sigma'\in$*futures* $\sigma$. $p\ \sigma'$
**by** (*simp add*: *state-property-decisions-def futures-def state-property-is-decided-def*)
**show** *state-properties-are-consistent* ($\bigcup\{$*state-property-decisions* $\sigma \mid \sigma. \sigma \in \sigma$*-set*$\}$)

**unfolding** *state-properties-are-consistent-def*
**by** ($metis$ ($mono\text{-}tags$, $lifting$) $\Sigma t\text{-}def$ $\langle \exists \sigma \in \Sigma t.\ \forall p \in \bigcup \{state\text{-}property\text{-}decisions\ \sigma \mid \sigma.\ \sigma \in \sigma\text{-}set\}.\ \forall \sigma' \in futures\ \sigma.\ p\ \sigma' \rangle$ $mem\text{-}Collect\text{-}eq$ $monotonic\text{-}futures$ $order\text{-}refl$)
**qed**

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

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

**lemma** (**in** *Protocol*) *naturally-corresponding-consistency* :
$\forall\ q\text{-}set.\ state\text{-}properties\text{-}are\text{-}consistent\ \{naturally\text{-}corresponding\text{-}state\text{-}property\ q \mid q.\ q \in q\text{-}set\}$
$\longrightarrow consensus\text{-}value\text{-}properties\text{-}are\text{-}consistent\ q\text{-}set$
**apply** (*rule*, *rule*)
**proof** −
**fix** *q-set*
**have**
$state\text{-}properties\text{-}are\text{-}consistent\ \{naturally\text{-}corresponding\text{-}state\text{-}property\ q \mid q.\ q \in q\text{-}set\}$
$\longrightarrow (\exists\ \sigma \in \Sigma.\ \forall\ p \in \{\lambda\sigma'.\ \forall\ c \in \varepsilon\ \sigma'.\ q\ c \mid q.\ q \in q\text{-}set\}.\ p\ \sigma)$
**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-set*. $q\ c)$
$\longrightarrow (\exists\ c \in C.\ \forall\ q' \in$ *q-set*. $q'\ c)$
**using** *is-valid-estimator-def ε-type* **by** *fastforce*
**ultimately show**
*state-properties-are-consistent* {*naturally-corresponding-state-property q* |*q*. *q* ∈ *q-set*}
⟹ *consensus-value-properties-are-consistent q-set*
**by** (*simp add*: *consensus-value-properties-are-consistent-def*)
**qed**

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

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

**theorem** (**in** *Protocol*) *n-party-safety-for-consensus-value-properties* :
∀ σ*-set*. σ*-set* ⊆ Σ*t*
⟶ *finite* σ*-set*
⟶ *is-faults-lt-threshold* (⋃ σ*-set*)
⟶ *consensus-value-properties-are-consistent* (⋃ {*consensus-value-property-decisions* σ | σ. σ ∈ σ*-set*})
**apply** (*rule*, *rule*, *rule*, *rule*)
**proof** −
**fix** σ*-set*
**assume** σ*-set* ⊆ Σ*t*
**and** *finite* σ*-set*
**and** *is-faults-lt-threshold* (⋃ σ*-set*)
**hence** *state-properties-are-consistent* (⋃ {*state-property-decisions* σ | σ. σ ∈ σ*-set*})
**using** ‹σ*-set* ⊆ Σ*t*› *n-party-safety-for-state-properties* **by** *auto*
**hence** *state-properties-are-consistent* {*p* ∈ ⋃ {*state-property-decisions* σ | σ. σ ∈ σ*-set*}. ∃ *q*. *p* = *naturally-corresponding-state-property q*}
**unfolding** *naturally-corresponding-state-property-def state-properties-are-consistent-def*
**apply** (*simp*)
**by** *meson*
**hence** *state-properties-are-consistent* {*naturally-corresponding-state-property q* | *q*. *naturally-corresponding-state-property q* ∈ ⋃ {*state-property-decisions* σ | σ. σ ∈ σ*-set*}}

**by** (*smt Collect-cong*)
**hence** *consensus-value-properties-are-consistent* $\{q.$ *naturally-corresponding-state-property* $q \in \bigcup$ $\{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}\}$
**using** *naturally-corresponding-consistency*
**proof** −
**show** *?thesis*
**by** (*metis* (*no-types*) *Setcompr-eq-image* ‹$\forall$ *q-set*. *state-properties-are-consistent* $\{$*naturally-corresponding-state-property* $q \mid q.\ q \in$ *q-set*$\}$ $\longrightarrow$ *consensus-value-properties-are-consistent* *q-set*› ‹*state-properties-are-consistent* $\{$*naturally-corresponding-state-property* $q \mid q.$ *naturally-corresponding-state-property* $q \in \bigcup\{$*state-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}\}$› *setcompr-eq-image*)
**qed**
**hence** *consensus-value-properties-are-consistent* ($\bigcup$ $\{$*consensus-value-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}$)
**apply** (*simp add*: *consensus-value-property-decisions-def consensus-value-property-is-decided-def state-property-decisions-def consensus-value-properties-are-consistent-def*)
**by** (*metis mem-Collect-eq*)
**thus**
*consensus-value-properties-are-consistent* ($\bigcup$ $\{$*consensus-value-property-decisions* $\sigma \mid \sigma.\ \sigma \in \sigma$*-set*$\}$)
**by** *simp*
**qed**

**end**
**theory** *SafetyOracle*

**imports** *Main CBCCasper LatestMessage StateTransition*

**begin**

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

**lemma** (**in** *Protocol*) *latest-justifications-from-non-equivocating-validators-type* :

$\forall\ \sigma\ v.\ \sigma \in \Sigma \wedge v \in V \longrightarrow$ *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* ($p$, $\sigma$) = $\{v \in$ *observed-non-equivocating-validators* $\sigma.\ \forall\ c \in$ *latest-estimates-from-non-equivocating-validators* $\sigma\ v.\ p\ c\}$

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

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

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

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

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

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

*weight-measure* (*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* ($p$, $m$, $v$, $\sigma$) = $\{m' \in$ *later-from* ($m$, $v$, $\sigma$). $\neg\ p$ (*est* $m'$)$\}$

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

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

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

**have** *later-from* ($m$,$v$,$\sigma$) = $\{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\}$
**apply** (*simp add*: *later-from-def from-sender-def later-def*)
**by** *auto*
**also have** ... = $\{m'' \in \sigma.$ *sender* $m'' = v \wedge$ *justified* $m\ m''\} \cup \emptyset$

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

**fun** (**in** *Params*) *gt-threshold* :: (*validator set* ∗ *state*) ⇒ *bool*
**where**
*gt-threshold* (*v-set*, $\sigma$)
= (*weight-measure v-set* > (*weight-measure v-set*) *div 2* + *t* − *weight-measure* (*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

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)


fun (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 σ b = sum W {v ∈ observed σ. ∃ b′ ∈ B. b′ ∈ (latest-estimates-from-non-equivocating-validators
σ v) ∧ (b ⇂ b′)}


definition (in GhostParams) children :: block ∗ state ⇒ block set
  where
    children = (λ(b, σ). {b′ ∈ est ‘σ. b = prev b′})


definition (in GhostParams) best-children :: block ∗ state ⇒  block set
  where
    best-children = (λ (b, σ). {arg-max-on (score σ) (children (b, σ))})
```

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

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

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

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

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

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

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

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

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

  **oops**

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

  **oops**

**end**