Mechanized Proofs For the Highway Protocol

Matthew Doty

November 4, 2020

Contents

theory *Highway*
  imports
    Main
    HOL−Lattice.Orders
    HOL.Rat
begin

sledgehammer-params [*smt-proofs* = *false*]

1 Axiomatization

Isabelle/HOL will not let us have a function *weight* ::*validator* ⇒ *rat* in a type class because it is missing a parameter. We use phantom parameters to get around this.

datatype *'a validator* = *validator nat*
datatype *'a vote-option* = *vote-for nat* | *abstain* (∅)

class *highway* = *partial-order* +
  fixes *vote-value* :: *'a* ⇒ *'a vote-option*
  fixes *sender* :: *'a* ⇒ *'a validator*
  fixes *weight* :: *'a validator* ⇒ *rat*
  assumes *non-neg-weights*: *weight v* ≥ *0*

2 Consistency

This is an elementary consistency proof, however it will be desirable to have model more complex sets of assumptions

datatype *example* = *example*

instantiation *example* :: *highway*
begin
fun *leq-example* :: *example* ⇒ *example* ⇒ *bool* **where**
  example ⊑ *example* = *True*

definition *vote-value-example* :: *example* ⇒ *example vote-option* **where**
  vote-value-example m = *vote-for 0*
definition *sender-example* :: *example* ⇒ *example validator* **where**
  sender-example m = *validator 0*
definition *weight-example* :: *example validator* ⇒ *rat* **where**
  weight-example v = *1*

instance
  apply (*standard*)
    apply (*metis* (*full-types*) *example.exhaust leq-example.simps*)
    apply (*metis* (*full-types*) *example.exhaust*)
    apply (*metis* (*full-types*) *example.exhaust*)
    apply (*simp add*: *weight-example-def*)
  done
end

3 Equivocations

definition (**in** *highway*)
  equivocation :: *'a validator* ⇒ *'a* ⇒ *'a* ⇒ *bool* **where**
  equivocation v x y ≡
      v = *sender x* ∧ *v* = *sender y* ∧ ¬ (*x* ⊑ *y*) ∧ ¬ (*y* ⊑ *x*)

lemma (**in** *highway*) *equivocation-neg*:
  assumes *v* = *sender x* **and** *v* = *sender y*
  shows ¬ *equivocation v x y* ≡ *x* ⊑ *y* ∨ *y* ⊑ *x*
  using *assms equivocation-def* **by** *auto*

abbreviation (**in** *partial-order*) *downset* :: *'a set* ⇒ *'a set* (↓) **where**
  ↓ *S* ≡ { *m* . ∃ *s* ∈ *S*. *m* ⊑ *s* }

lemma (**in** *partial-order*) *downset-universe*:
  ↓ (*UNIV* :: *'a set*) = *UNIV*
  using *leq-refl*
  by *fastforce*

definition (**in** *highway*)

byzantine-in :: *'a set* ⇒ *'a validator* ⇒ *bool* **where**
byzantine-in S v ≡ ∃ *x* ∈ ↓ *S*. ∃ *y* ∈ ↓ *S*. *equivocation v x y*

definition (**in** *highway*) *byzantine* :: *'a validator* ⇒ *bool* **where**
byzantine v ≡ *byzantine-in UNIV v*

lemma *byzantine-def'*:
byzantine v = (∃ *x y*. *equivocation v x y*)
unfolding *byzantine-def byzantine-in-def*
by (*metis* (*mono-tags*, *lifting*) *UNIV-I downset-universe*)

definition (**in** *highway*)
honest-message-weight :: *'a set* ⇒ *'a set* ⇒ *rat* ($weight_{\mathcal{M}}$) **where**
$weight_{\mathcal{M}}$ *S T* =
(∑ *v* | (∃ *s* ∈ *T* . *v* = *sender s*)
∧ *weight v* ≠ *0*
∧ ¬ *byzantine-in S v*
. *weight v*)

lemma (**in** *highway*) *messages-weight-mono*:
assumes
T ⊆ *U*
finite {*v*. ∃ *s* ∈ *U*. *v* = *sender s* ∧ *weight v* ≠ *0* ∧ ¬ *byzantine-in S v*}
shows $weight_{\mathcal{M}}$ *S T* ≤ $weight_{\mathcal{M}}$ *S U*
proof −
have {*v*. ∃ *s* ∈ *T*. *v* = *sender s* ∧ *weight v* ≠ *0* ∧ ¬ *byzantine-in S v*}
⊆ {*v*. ∃ *s* ∈ *U*. *v* = *sender s* ∧ *weight v* ≠ *0* ∧ ¬ *byzantine-in S v*}
using *assms*(*1*)
by *blast*
with *assms*(*2*) **show** *?thesis*
unfolding *honest-message-weight-def*
by (*simp add*: *non-neg-weights sum-mono2*)
qed

definition (**in** *highway*)
most-recent-for :: *'a validator* ⇒ *'a set* ⇒ *'a set* **where**
most-recent-for v S =
{ *s* ∈ *S* . *v* = *sender s*
∧ (∀ *t* ∈ *S*. *v* = *sender t* ⟶ *s* ⊑ *t* ⟶ *s* = *t*) }

lemma (**in** *highway*) *most-recent-for-idem*:
most-recent-for v (*most-recent-for v S*) = *most-recent-for v S*
unfolding *most-recent-for-def* **by** *fastforce*

fun (**in** *partial-order*) *descending-chain-list* :: *'a list* ⇒ *bool* **where**
descending-chain-list [] = *True*
| *descending-chain-list* [*x*] = *True*
| *descending-chain-list* (*x1* # *x2* # *xs*)
= (*x2* ⊑ *x1* ∧ *x2* ≠ *x1* ∧ *descending-chain-list* (*x2* # *xs*))

```
lemma (in partial-order) descending-chain-list-drop-penultimate:
  descending-chain-list (x1 # x2 # xs) ⟹ descending-chain-list (x1 # xs)
  by (induct xs,
        simp,
        metis descending-chain-list.simps(3) leq-antisym leq-trans)

lemma (in partial-order) descending-chain-list-greater-than-others:
  assumes descending-chain-list (x # xs)
  shows ∀ y ∈ set xs . y ⊑ x ∧ y ≠ x
  using assms descending-chain-list-drop-penultimate
  by (induct xs, fastforce+)

lemma (in partial-order) descending-chain-list-distinct:
  descending-chain-list xs ⟹ distinct xs
  by (induct xs,
        simp,
        metis
          distinct.simps(2)
          descending-chain-list.elims(3)
          descending-chain-list.simps(3)
          descending-chain-list-greater-than-others)

lemma (in highway) most-recent-exists:
  assumes finite S m ∈ S v = sender m
  shows ∃ n ∈ most-recent-for v S . m ⊑ n
proof (rule ccontr)
  assume ¬ (∃ n ∈ most-recent-for v S . m ⊑ n)
  hence fresh:
    ∀ n ∈ S. m ⊑ n
        ⟶ v = sender n
        ⟶ (∃ p ∈ S. v = sender p ∧ n ≠ p ∧ m ⊑ n ∧ n ⊑ p)
    using most-recent-for-def by auto
  {
    fix n :: nat
    have ∃ xs .
            descending-chain-list xs
            ∧ length xs = n
            ∧ set xs ⊆ S
            ∧ (∀ x ∈ set xs. v = sender x ∧ m ⊑ x)
    proof (induct n)
      case 0
      then show ?case
        by simp
    next
      case (Suc n)
      then show ?case
      proof (cases n = 0)
        assume n = 0
```

```
    have
      descending-chain-list [m]
      length [m] = Suc 0
      set [m] ⊆ S
      ∀ x ∈ set [m] . v = sender x ∧ m ⊑ x
      using assms leq-refl
      by auto
    thus ?case
      unfolding ‹n = 0›
      by blast
  next
    assume
      n ≠ 0
      ∃xs.
         descending-chain-list xs
         ∧ length xs = n
         ∧ set xs ⊆ S
         ∧ (∀x∈set xs. v = sender x ∧ m ⊑ x)
    from this obtain x xs where
        descending-chain-list (x # xs)
        length (x # xs) = n
        set (x # xs) ⊆ S
        ∀ x' ∈ set (x # xs). v = sender x' ∧ m ⊑ x'
        m ⊑ x
      by (metis
            (no-types, lifting)
            length-0-conv
            length-greater-0-conv
            descending-chain-list.elims(2)
            nth-Cons-0 nth-mem)
    moreover from this obtain y where
        y ∈ S
        v = sender y
        y ≠ x
        x ⊑ y
        m ⊑ y
      by (metis fresh list.set-intros(1) leq-trans subset-eq)
    ultimately have
        descending-chain-list (y # x # xs)
        length (y # x # xs) = Suc n
        set (y # x # xs) ⊆ S
        ∀ x' ∈ set (y # x # xs). v = sender x' ∧ m ⊑ x'
      by auto
    thus ?case by blast
  qed
 qed
}
from this obtain xs :: 'a list where
  descending-chain-list xs
```

length xs = *card S* + *1*
set xs ⊆ *S*
by *blast*
with ‹*finite S*› ‹*descending-chain-list xs*› ‹*set xs* ⊆ *S*›
have *length xs* ≤ *card S*
by (*metis card-mono distinct-card descending-chain-list-distinct*)
with ‹*length xs* = *card S* + *1*› **show** *False*
by *linarith*
qed

lemma (**in** *highway*) *non-byzantine-most-recent-for*:
assumes ¬ *byzantine-in S v*
and *m* ∈ *most-recent-for v S*
and *n* ∈ *most-recent-for v S*
shows *m* = *n*
proof −
have *m* ∈ *S n* ∈ *S v* = *sender m v* = *sender n*
using *assms most-recent-for-def* **by** *auto*
moreover with *assms*(*1*) **have** *m* ⊑ *n* ∨ *n* ⊑ *m*
unfolding *byzantine-in-def equivocation-def*
using *leq-refl* **by** *blast*
ultimately show *?thesis*
using *assms*(*2*) *assms*(*3*) *most-recent-for-def* **by** *force*
qed

lemma (**in** *highway*) *most-recent-exists-unique*:
assumes
finite S
¬ *byzantine-in S v*
m ∈ *S*
v = *sender m*
shows ∃! *n* ∈ *most-recent-for v S* . *m* ⊑ *n*
proof (*rule ccontr*)
assume ¬ (∃! *n* . *n* ∈ *most-recent-for v S* ∧ *m* ⊑ *n*)
moreover have ∃ *n*. *n* ∈ *most-recent-for v S* ∧ *m* ⊑ *n*
using *assms*(*1*) *assms*(*3*) *assms*(*4*) *most-recent-exists* **by** *blast*
ultimately obtain *p q* **where**
p ∈ *most-recent-for v S*
q ∈ *most-recent-for v S*
p ≠ *q*
by *blast*
thus *False*
using *assms*(*2*) *non-byzantine-most-recent-for*
by *blast*
qed

definition (**in** *highway*) *most-recent* :: *'a set* ⇒ *'a set* **where**
most-recent S = ⋃{ *s* . ∃ *v*. *s* = *most-recent-for v S* }

lemma (**in** *highway*) *most-recent-idem*:
  most-recent (*most-recent* S) = *most-recent* S
  unfolding *most-recent-def most-recent-for-def* **by** *auto*

lemma (**in** *highway*) *downset-most-recent-for-subset*:
  $\downarrow$ (*most-recent* S) $\subseteq \downarrow S$
proof (*intro subsetI*)
  fix x
  assume $x \in \downarrow$ (*most-recent* S)
  hence $\exists\ y \in S\ .\ x \sqsubseteq y$
    unfolding *most-recent-def most-recent-for-def*
    by *blast*
  thus $x \in \downarrow S$ **by** *auto*
qed

lemma (**in** *highway*) *downset-most-recent-for*:
  assumes *finite* S
  shows $\downarrow$ (*most-recent* S) $= \downarrow S$
proof (*intro equalityI subsetI*)
  fix x
  assume $x \in \downarrow$ (*most-recent* S)
  thus $x \in \downarrow S$
    using *downset-most-recent-for-subset* **by** *blast*
next
  fix x
  assume $x \in \downarrow S$
  from *this* **obtain** v y m **where**
      v = *sender* y
      $m \in$ *most-recent-for* v S
      $x \sqsubseteq y$
      $y \sqsubseteq m$
    using ⟨*finite* S⟩ *most-recent-exists* **by** *fastforce*
  hence $x \in \downarrow$ (*most-recent-for* v S)
    by (*metis CollectI leq-trans*)
  thus $x \in \downarrow$ (*most-recent* S)
    unfolding *most-recent-def*
    by *blast*
qed

lemma (**in** *highway*) *most-recent-byzantine-impl*:
  byzantine-in (*most-recent* S) $v \Longrightarrow$ *byzantine-in* S v
  unfolding
    most-recent-def
    most-recent-for-def
    byzantine-in-def
    equivocation-def
  by *blast*

lemma (**in** *highway*) *most-recent-byzantine-iff*:

assumes *finite S*
shows *byzantine-in* (*most-recent S*) v = *byzantine-in S v*
proof
assume *byzantine-in* (*most-recent S*) v
thus *byzantine-in S v*
unfolding
most-recent-def
most-recent-for-def
byzantine-in-def
equivocation-def
by *blast*
next
assume *byzantine-in S v*
from *this* **obtain** $x\ y\ p\ q\ t\ u$ **where**
$p \in S$
$q \in S$
$t = sender\ p$
$u = sender\ q$
$x \sqsubseteq p$
$y \sqsubseteq q$
$v = sender\ x$
$v = sender\ y$
$\neg\ x \sqsubseteq y$
$\neg\ y \sqsubseteq x$
unfolding *byzantine-in-def equivocation-def*
by *blast*
moreover from *this* **obtain** $m\ n$ **where**
$p \sqsubseteq m$
$q \sqsubseteq n$
$m \in$ *most-recent S*
$n \in$ *most-recent S*
unfolding *most-recent-def*
using
most-recent-exists [*OF* ⟨*finite S*⟩ ⟨$p \in S$⟩ ⟨$t = sender\ p$⟩]
most-recent-exists [*OF* ⟨*finite S*⟩ ⟨$q \in S$⟩ ⟨$u = sender\ q$⟩]
by *blast*
ultimately
show *byzantine-in* (*most-recent S*) v
unfolding *byzantine-in-def equivocation-def*
using *leq-trans* **by** *blast*
qed

definition (**in** *highway*)
majority-option :: *'a set* ⇒ *'a vote-option* ⇒ *bool* **where**
majority-option S a =
($weight_{\mathcal{M}}$ *S* { $s \in$ *most-recent S* . a = *vote-value s* }
> $weight_{\mathcal{M}}$ *S* { $s \in$ *most-recent S* . $a \neq$ *vote-value s* })

lemma (**in** *highway*) *at-most-one-majority-option*:

assumes *finite S majority-option S v* **and** *majority-option S w*
shows $v = w$
proof (*rule ccontr*)
let $?S' = \textit{most-recent } S$
assume $v \neq w$
hence
$\{\ s \in ?S'\ .\ v = \textit{vote-value } s\ \} \subseteq \{\ s \in ?S'\ .\ w \neq \textit{vote-value } s\ \}$
(**is** $?v\text{-}votes \subseteq ?w\text{-}opposing\text{-}votes$)
$\{\ s \in ?S'\ .\ w = \textit{vote-value } s\ \} \subseteq \{\ s \in ?S'\ .\ v \neq \textit{vote-value } s\ \}$
(**is** $?w\text{-}votes \subseteq ?v\text{-}opposing\text{-}votes$)
by *blast+*
moreover have $?S' \subseteq S$
unfolding *most-recent-def most-recent-for-def*
by *blast*
hence *finite* $?S'$
using *assms(1) infinite-super* **by** *auto*
ultimately have
$\textit{weight}_{\mathcal{M}}\ S\ ?v\text{-}votes \leq \textit{weight}_{\mathcal{M}}\ S\ ?w\text{-}opposing\text{-}votes$
$\textit{weight}_{\mathcal{M}}\ S\ ?w\text{-}votes \leq \textit{weight}_{\mathcal{M}}\ S\ ?v\text{-}opposing\text{-}votes$
by (*simp add*: *messages-weight-mono*)+
hence
$\textit{weight}_{\mathcal{M}}\ S\ ?v\text{-}votes < \textit{weight}_{\mathcal{M}}\ S\ ?w\text{-}votes$
$\textit{weight}_{\mathcal{M}}\ S\ ?w\text{-}votes < \textit{weight}_{\mathcal{M}}\ S\ ?v\text{-}votes$
using *assms*
unfolding *majority-option-def*
by *linarith+*
thus *False*
by *auto*
qed

4 Finality

class *highway-summit* = *highway* +
assumes *finite-weight*: $\textit{finite}\ \{\ v\ .\ \textit{weight}\ v \neq 0\ \}$
assumes *finite-citations*: $\textit{finite}\ (\downarrow \{\ m\ \})$
assumes *majority-driven*:
$\textit{majority-option}\ \{u\ .\ u \sqsubseteq m \wedge u \neq m\}\ a \Longrightarrow \textit{vote-value}\ m = a$

definition (**in** *highway*)
$\textit{horizon} :: {}'a\ \textit{set} \Rightarrow {}'a\ \textit{vote-option} \Rightarrow {}'a\ \textit{set} \Rightarrow \textit{bool}$ **where**
$\textit{horizon state a x} \equiv$
$x \neq \{\} \wedge x \subseteq \downarrow \textit{state}$
$\wedge\ (\forall\ m \in x.\ \forall\ n \in \downarrow \textit{state}.$
$\textit{sender } m = \textit{sender } n$
$\longrightarrow m \sqsubseteq n$
$\longrightarrow \textit{vote-value } n = a)$

definition (**in** *highway*) $\textit{validator-weight} :: ({}'a\ \textit{validator})\ \textit{set} \Rightarrow \textit{rat}$ **where**
[*simp*]: $\textit{validator-weight}\ V = (\sum\ v \mid v \in V \wedge \textit{weight}\ v \neq 0\ .\ \textit{weight}\ v)$

```
definition (in highway) committee-weight
  :: 'a ⇒ 'a set ⇒ 'a set ⇒ rat where
  [simp]: committee-weight m y x =
            validator-weight
              { v. ∃ u ∈ y. v = sender u
                   ∧ (∃ n ∈ x. v = sender n ∧ n ⊑ u ∧ n ⊑ m) }

fun (in highway) summit
   :: rat ⇒ 'a set ⇒ 'a vote-option ⇒ ('a set) list ⇒ bool where
    summit - - - [] = True
  | summit - state a [x] = horizon state a x
  | summit q state a (x # x' # xs) =
      (horizon state a x
         ∧ (∀ m ∈ x. q ≤ committee-weight m x x')
         ∧ summit q state a (x' # xs))

lemma (in highway) summit-left-weaken:
  summit q state a (xs @ ys) ⟹ summit q state a xs
proof (induct xs)
  case Nil
  then show ?case by simp
next
  case (Cons x xs)
  then show ?case
  proof (cases xs = [])
    case True
    show ?thesis
      by (metis
            True
            Cons.prems
            append-Cons
            summit.simps(2)
            summit.simps(3)
            min-list.cases)
  next
    case False
    from this obtain x' xs' where
      xs = x' # xs'
      by (meson neq-Nil-conv)
    then show ?thesis
      using Cons.hyps Cons.prems
      unfolding summit.simps(3) by auto
  qed
qed

definition (in highway) message-weight :: 'a set ⇒ rat where
  [simp]: message-weight M = validator-weight (image sender M)
```

theorem (**in** *highway-summit*) *elementary-finality*:
  assumes
    summit (*validator-weight UNIV / 2*) *S a* [*y*,*x*]
    validator-weight { *v*. *byzantine v* } = *0*
    $y \subseteq \downarrow \{n\}$
  shows *vote-value n* = *a*
  sorry

end