SP

nano

November 17, 2018

Contents

theory *ImplicitQuorums*
imports *Main HOL−Eisbach.Eisbach*
begin

1 personal quorums

locale *personal* =
fixes $quorums :: 'node \Rightarrow 'node\ set\ set$ **and** $W::'node\ set$
assumes $p1:\bigwedge p\ .\ Q \in quorums\ p \Longrightarrow p \in Q$
and $p2:\bigwedge p\ p'\ Q\ .\ [\![Q \in quorums\ p;\ p' \in Q]\!] \Longrightarrow Q \in quorums\ p'$
— the existence of at least one quorum per participant and closure under union are unnecessary for what follows
begin

definition *is-intact* **where**
is-intact $I \equiv I \subseteq W \wedge (\forall\ p \in I\ .\ \exists\ Q \in$ *quorums* $p\ .\ Q \subseteq I)$
$\wedge\ (\forall\ p\ p'\ Q\ Q'\ .\ p \in I \wedge p' \in I \wedge Q \in$ *quorums* $p \wedge Q' \in$ *quorums* $p' \longrightarrow W \cap Q \cap Q' \neq \{\})$

definition *blocked* **where**
blocked $R\ p \equiv \forall\ Q \in$ *quorums* $p\ .\ Q \cap R \neq \{\}$

lemma $\bigwedge q$. *blocked* $\{p$. *blocked* $R\ p\}\ q \Longrightarrow$ *blocked* $R\ q$
using *p2 p1* **unfolding** *blocked-def* **by** *fastforce*

lemma *quorum-not-empty*:
assumes $q \in$ *quorums* n
shows $q \neq \{\}$
by (*metis assms empty-iff personal-axioms personal-def*)

lemma *intact-union*:
assumes *is-intact* I_1 **and** *is-intact* I_2 **and** $I_1 \cap I_2 \neq \{\}$
shows *is-intact* $(I_1 \cup I_2)$
proof −
have $I_1 \cup I_2 \subseteq W$
using *assms*(*1*) *assms*(*2*) *is-intact-def* **by** *auto*
moreover
have $\exists\ Q \in$ *quorums* $p\ .\ Q \subseteq I_1 \cup I_2$ **if** $p \in I_1 \cup I_2$ **for** p
using ‹*is-intact* I_1› ‹*is-intact* I_2› *that* **unfolding** *is-intact-def*
by (*meson Un-iff le-supI1 sup.coboundedI2*)
moreover
have $W \cap q_1 \cap q_2 \neq \{\}$
if $n_1 \in I_1 \cup I_2$ **and** $q_1 \in$ *quorums* n_1 **and** $n_2 \in I_1 \cup I_2$ **and** $q_2 \in$ *quorums* n_2 **for** $q_1\ q_2\ n_1\ n_2$
proof −
have ‹$W \cap q_1 \cap q_2 \neq \{\}$›
if $n_1 \in I$ **and** $n_2 \in I$ **and** $I = I_1 \vee I = I_2$ **for** I
using ‹$q_1 \in$ *quorums* n_1 › ‹$q_2 \in$ *quorums* n_2› *assms*(*1*,*2*) *that* **unfolding** *is-intact-def*
by (*metis Int-commute inf-left-commute*)
moreover
have ‹$W \cap q_1 \cap q_2 \neq \{\}$›
if $n_1 \in I_1$ **and** $n_2 \in I_2$ **and** *is-intact* I_1 **and** *is-intact* I_2
and $q_2 \in$ *quorums* n_2 **and** $q_1 \in$ *quorums* n_1 **and** $I_1 \cap I_2 \neq \{\}$
for $q_1\ q_2\ n_1\ n_2\ I_1\ I_2$ — We generalize to avoid repeating the argument twice
proof −
obtain n_3 **where** $n_3 \in I_1 \cap I_2$ **using** ‹$I_1 \cap I_2 \neq \{\}$› **by** *blast*
obtain q_3 **where** $q_3 \in$ *quorums* n_3 **and** $q_3 \subseteq I_1$
using ‹*is-intact* I_1› ‹$n_3 \in I_1 \cap I_2$› *is-intact-def* **by** *fastforce*
moreover
have $q_3 \cap q_2 \neq \{\}$
by (*metis IntD2 Int-assoc Int-empty-right* ‹$n_3 \in I_1 \cap I_2$› *calculation*(*1*)

is-intact-def that(2) that(4) that(5))
ultimately
obtain n_4 **where** $n_4 \in I_1$ **and** $q_2 \in$ *quorums* n_4
by (*meson Int-emptyI* ‹$q_2 \in$ *quorums* n_2› *personal-axioms personal-def subset-iff*)
thus $W \cap q_1 \cap q_2 \neq \{\}$ **using** ‹*is-intact* I_1› ‹$q_1 \in$ *quorums* n_1› ‹$n_1 \in I_1$›
unfolding *is-intact-def* **by** *blast*
qed
ultimately show *?thesis* **using** *that assms* **by** *fast*
qed
ultimately show *?thesis* **unfolding** *is-intact-def* **by** *fastforce*
qed

end

2 slices

locale *slices* =
fixes *slices* :: *'node* ⇒ *'node set set* — the quorum slices

~~$\bigwedge n\ .\ slices\ n \neq \{\} \wedge (\forall\ S \in slices\ n\ .\ n \notin S)$~~
~~and $\bigwedge p\ S\ S'\ .\ \neg(S \in slices\ p \wedge S' \in slices\ p \wedge S \subset S')$ — ‹The last assumption makes sense if one thinks of slices in terms of trust.›~~
begin

definition *quorum* **where**
$quorum\ q \equiv \forall\ n \in q\ .\ \exists\ S \in slices\ n\ .\ S \subseteq q$

definition *quorum-of* **where**
$quorum\text{-}of\ n\ q \equiv n \in q \wedge quorum\ q$

lemma *quorums-closed*:
assumes *quorum-of* n q **and** $n' \in q$
shows *quorum-of* n' q
using *assms* **unfolding** *quorum-of-def* **by** *auto*

lemma *quorum-union*:
assumes *quorum* q_1 **and** *quorum* q_2
shows *quorum* $(q_1 \cup q_2)$ **using** *assms* **unfolding** *quorum-def*
by (*meson UnE le-supI1 sup.coboundedI2 sup-eq-bot-iff*)

definition *quorum-blocking* **where**
$quorum\text{-}blocking\ B\ p \equiv \forall\ Q\ .\ quorum\text{-}of\ p\ Q \longrightarrow Q \cap B \neq \{\}$

inductive *blocking* **where**
$p \in R \Longrightarrow blocking\ R\ p$
$|\ \forall\ Sl \in slices\ p\ .\ \exists\ q \in Sl\ .\ blocking\ R\ q \Longrightarrow blocking\ R\ p$

2.1 *blocking* and *quorum-blocking* are equivalent

lemma *not-blocking*:$\neg$*blocking R p* $\Longrightarrow$ *p* $\notin$ *R* $\wedge$ ($\exists$ *Sl* $\in$ *slices p* . $\forall$ *q* $\in$ *Sl* . $\neg$ *blocking R q*)
  by (*meson blocking.simps*)

inductive *not-blocked* **for** *p R* **where**
  $[\![$*p* $\notin$ *R*; *Sl* $\in$ *slices p*; $\forall$ *q* $\in$ *Sl* . $\neg$ *blocking R q*$]\!]$ $\Longrightarrow$ *not-blocked p R p*
| $[\![$*not-blocked p R p'*; *Sl* $\in$ *slices p'*; $\forall$ *q* $\in$ *Sl* . $\neg$ *blocking R q*; *p''* $\in$ *Sl*$]\!]$ $\Longrightarrow$ *not-blocked p R p''*

lemma *ne-not-blocked-is-quorum*:
  fixes *Q p R*
  defines *Q* $\equiv$ {*q* . *not-blocked p R q*}
  assumes *Q* $\neq$ {}
  shows *quorum Q*
proof –
  have $\forall$ *n*$\in$*Q* . $\exists$ *S*$\in$*slices n* . *S*$\subseteq$*Q*
  proof (*simp add*: *Q-def*; *clarify*)
    fix *n*
    assume *not-blocked p R n*
    thus $\exists$*S*$\in$*slices n*. *S* $\subseteq$ *Collect* (*not-blocked p R*)
    proof (*cases*)
      case (*1 Sl*)
      then show *?thesis*
        by (*metis* (*full-types*) *Ball-Collect not-blocked.intros*)
    next
      case (*2 p' Sl*)
      hence $\neg$*blocking R n* **by** *simp*
       with *this* **obtain** *Sl* **where** *n* $\notin$ *R* **and** *Sl* $\in$ *slices n* **and** $\forall$ *q* $\in$ *Sl* . $\neg$ *blocking R q*
        by (*meson blocking.intros*(*2*) *blocking.intros*(*1*))
      moreover note ‹*not-blocked p R n*›
     ultimately show *?thesis* **by** (*metis* (*full-types*) *Ball-Collect not-blocked.intros*(*2*))
    qed
  qed
  thus *?thesis*
    by (*simp add*: *assms*(*2*) *quorum-def*)
qed

lemma *not-blocked-disjoint-R*:
  fixes *Q p R*
  defines *Q* $\equiv$ {*q* . *not-blocked p R q*}
  shows *Q* $\cap$ *R* = {}
proof –
  have *q* $\notin$ *R* **if** *not-blocked p R q* **for** *q*
    using *that*
  proof (*induct*)
    case (*1 Sl*)
    then show *?case* **by** *auto*

```
  next
    case (2 p' Sl p'')
    then show ?case using blocking.intros(1) by blast
  qed
  thus ?thesis unfolding Q-def by auto
qed

lemma quorum-blocking-blocking:
  assumes quorum-blocking R p shows blocking R p
proof −
  have ¬quorum-blocking R p if ¬blocking R p
    — this is the contrapositive
  proof −
    define Q where Q ≡ {q . not-blocked p R q}
    have p ∈ Q using ‹¬blocking R p›
      by (metis Q-def mem-Collect-eq not-blocked.intros(1) not-blocking)
    moreover
    have quorum Q
      using Q-def ‹p ∈ Q› ne-not-blocked-is-quorum by auto
    moreover
    have Q ∩ R = {}
      by (simp add: Q-def not-blocked-disjoint-R)
     ultimately show ?thesis unfolding quorum-blocking-def quorum-of-def by
blast
  qed
  thus ?thesis using assms by auto
qed

lemma quorum-is-quorum-of-some-slice:
  assumes quorum-of p Q
  obtains S where S ∈ slices p and S ⊆ Q
    and ⋀ p' . p' ∈ S ⟹ quorum-of p' Q
  using assms unfolding quorum-of-def quorum-def
  by (meson rev-subsetD)

lemma blocking-imp-quorum-blocking:
  assumes blocking R p shows quorum-blocking R p
  using assms
proof (induct)
case (1 p R)
  then show ?case
    using quorum-blocking-def quorum-of-def by auto
next
  case (2 p R)
  then show ?case unfolding quorum-blocking-def
    by (meson quorum-is-quorum-of-some-slice)
qed

lemma blocking-eq-quorum-blocking:
```

blocking R p = quorum-blocking R p
using *blocking-imp-quorum-blocking quorum-blocking-blocking* **by** *blast*

~~subsection old stuffdefinition slice-blocking where slice-blocking B p ≡ p ∈ B ∨ (∀ S ∈ slices p . S ∩ B ≠ {})lemma cascade: — ‹What is this?› assumes p ∉ B and quorum-blocking B p shows ∃ p' ∈ −B . slice-blocking B p' proof (rule ccontr; auto) assume ∀ p' ∈ −B . ¬slice-blocking B p' hence quorum (−B) unfolding quorum-blocking-def slice-blocking-def quorum-def quorum-of-def using ‹p ∉ B› disjoint-eq-subset-Compl by fastforce with ‹p ∉ B› and ‹quorum-blocking B p› show False using quorum-blocking-def quorum-of-def by auto qedlemma l:⟦¬blocking R p; not-blocked p R p'⟧ ⟹ ¬blocking R p' using not-blocked.cases by blast~~
end

3 projection

locale *projection = slices +*
fixes $W :: 'a\ set$ — this is the set on which we project the system
begin

definition *proj-slices* **where**
— slices projected on the well-behaved participants
proj-slices p ≡ {$S \cap W$ | S . $S \in$ *slices p*}

Now we instantiate the slices locale using the projected slices

interpretation *proj*: *slices proj-slices* **.**

lemma *quorum-is-proj-quorum*:
assumes *quorum q* **shows** *proj.quorum q*
unfolding *proj.quorum-def*
proof −
have ∃ *S*∈*proj-slices n*. *S* ⊆ *q* **if** *n* ∈ *q* **for** *n*
proof −
have ∃ *S*∈*slices n*. *S* ⊆ *q* **if** *n* ∈ *q* **for** *n* **using** *assms that* **unfolding** *quorum-def* **by** *auto*
moreover
have ∃ *S'*∈ *proj-slices n* . *S'* ⊆ *S* **if** *S*∈*slices n* **for** *S* **unfolding** *proj-slices-def* **using** *that* **by** *auto*
ultimately show *?thesis*
by (*meson order.trans that*)
qed
thus ∀ *n*∈*q*. ∃ *S*∈*proj-slices n*. *S* ⊆ *q*
by *blast*
qed

lemma *proj-blocking-is-blocking*:
assumes *proj.quorum-blocking B p*
shows *quorum-blocking B p*
by (*meson assms quorum-is-proj-quorum slices.quorum-blocking-def slices.quorum-of-def*)

~~lemma proj-blocking-is-blocking: assumes quorum-blocking B p and B ∩ W ≠ {} and p ∉ W shows proj.quorum-blocking B p nitpick[card 'a=3] oopslemma W-slice-blocking-is-proj-slice-blocking: slice-blocking (U ∩ W) n = proj.slice-blocking (U ∩ W) n unfolding proj.slice-blocking-def proj-slices-def slice-blocking-def by auto~~

definition *proj-of* **where**
  $\textit{proj-of } q \equiv \{p \in q \cap W \ . \ \exists \ S \in \textit{slices } p \ . \ S \cap W \subseteq q\}$

lemma *proj-is-intersection*:
  assumes *quorum q* **shows** $\textit{proj-of } q = q \cap W$
  using *assms* **unfolding** *quorum-def proj-of-def* **apply** *auto*
  using *inf.absorb-iff2* **by** *fastforce*

lemma *l3*: — needed?
  assumes $S \subseteq Q \cap W$ **and** *quorum Q*
  shows $S \subseteq \textit{proj-of } Q$
  using *assms* **unfolding** *quorum-def proj-of-def*
  using *Ball-Collect subset-eq* **by** *fastforce*

lemma *proj-of-is-proj-quorum*:
  assumes *quorum q* **shows** *proj.quorum* (*proj-of q*)
  using *assms* **unfolding** *proj.quorum-def quorum-def proj-slices-def*
  by (*simp add*: *proj-is-intersection*[*OF assms*(*1*)]; *meson Int-commute Int-iff inf-le1 inf-le2 subset-trans*)

lemma *quorum-in-W-is-proj-of*:
  assumes *quorum q* **and** $q \subseteq W$ **shows** $\textit{proj-of } q = q$
  using *assms* **unfolding** *quorum-def proj-of-def*
  by (*auto*; *metis inf-absorb1 order.trans*)

4 pseudo-quorums

definition *pseudo-quorum* **where**
  $\textit{pseudo-quorum } Q \equiv \forall \ p \in Q \cap W \ . \ \exists \ \textit{Sl} \in \textit{slices } p \ . \ \textit{Sl} \subseteq Q$

inductive *reachable-in-W* **for** *p Q* **where**
  $p \in Q \cap W \Longrightarrow \textit{reachable-in-W } p \ Q \ p$
| $[\![\textit{reachable-in-W } p \ Q \ p'; \ S \in \textit{slices } p'; \ S \subseteq Q; \ p'' \in S \cap W]\!] \Longrightarrow \textit{reachable-in-W } p \ Q \ p''$

lemma *p-in-reachable-from-p*:
  fixes *p Q*
  defines $Q' \equiv \{p' \ . \ \textit{reachable-in-W } p \ Q \ p'\}$
  assumes $p \in Q \cap W$
  shows $p \in Q'$
  using *Q′-def assms*(*2*) *reachable-in-W.intros*(*1*) **by** *auto*

lemma *reachable-from-p-subset-W*:
fixes $p\ Q$
defines $Q' \equiv \{p'\ .\ \textit{reachable-in-W}\ p\ Q\ p'\}$
assumes $p \in W$
shows $Q' \subseteq Q \cap W$ **unfolding** *Q′-def*
proof (*clarify*)
fix p'
assume *reachable-in-W* $p\ Q\ p'$
thus $p' \in Q \cap W$ **by** (*induct*; *auto*)
qed

lemma *pseudo-quorum-contains-proj-quorum*:
fixes $p\ Q$
defines $Q' \equiv \{p'\ .\ \textit{reachable-in-W}\ p\ Q\ p'\}$
assumes *pseudo-quorum* Q **and** $p \in Q \cap W$
shows *proj.quorum* Q'
unfolding *proj.quorum-def*
proof −
show $\forall\, n{\in}Q'.\ \exists\, S{\in}\textit{proj-slices}\ n.\ S \subseteq Q'$
unfolding *Q′-def*
proof (*simp*;*clarify*)
fix n
assume *reachable-in-W* $p\ Q\ n$
thus $\exists\, S{\in}\textit{proj-slices}\ n.\ S \subseteq \textit{Collect}\ (\textit{reachable-in-W}\ p\ Q)$
proof (*cases*)
case *1*
with ‹*pseudo-quorum* Q› **obtain** S **where** $S \in \textit{slices}\ n$ **and** $S \subseteq Q$
by (*meson assms*(*3*) *pseudo-quorum-def*)
hence *reachable-in-W* $p\ Q\ p'$ **if** $p' \in S \cap W$ **for** p'
by (*meson* ‹*reachable-in-W* $p\ Q\ n$› *that reachable-in-W.intros*(*2*))
thus *?thesis* **unfolding** *proj-slices-def* **using** ‹$S \in \textit{slices}\ n$› **by** *auto*
next
case (*2* $p'\ S$)
obtain S' **where** $S' \in \textit{slices}\ n$ **and** $S' \subseteq Q$
by (*meson 2*(*3*) *2*(*4*) *Int-iff assms*(*2*) *pseudo-quorum-def subset-iff*)
hence *reachable-in-W* $p\ Q\ p'$ **if** $p' \in S' \cap W$ **for** p'
using ‹*reachable-in-W* $p\ Q\ n$› *reachable-in-W.simps that* **by** *blast*
thus *?thesis* **unfolding** *proj-slices-def* **using** ‹$S' \in \textit{slices}\ n$› **by** *auto*
qed
qed
qed

definition *intertwined* **where**
intertwined $S \equiv \forall\ n \in S\ .\ \forall\ n' \in S\ .\ \forall\ q\ q'\ .$
proj.quorum-of $n\ q \wedge$ *proj.quorum-of* $n'\ q' \longrightarrow q \cap q' \cap W \neq \{\}$

lemma *pseudo-quorum-intersection*:
assumes *intertwined* S **and** $S \subseteq W$ **and** *pseudo-quorum* Q **and** *pseudo-quorum* Q' **and** $p \in S{\cap}Q$ **and** $p' \in S{\cap}Q'$

shows $Q \cap Q' \cap W \neq \{\}$
proof −
have $p \in Q \cap W$ **and** $p' \in Q' \cap W$
using *IntD2 IntI assms*(*2*,*5*,*6*) **by** *auto*
with *this* **obtain** *Q-proj* **and** *Q′-proj* **where** *proj.quorum Q-proj* **and** *Q-proj* $\subseteq Q \cap W$ **and** $p \in$ *Q-proj*
and *proj.quorum Q′-proj* **and** *Q′-proj* $\subseteq Q' \cap W$ **and** $p' \in$ *Q′-proj*
using *pseudo-quorum-contains-proj-quorum p-in-reachable-from-p reachable-from-p-subset-W* ‹*pseudo-quorum Q*› ‹*pseudo-quorum Q′*›
by (*auto*; *metis*)
have *Q-proj* $\cap$ *Q′-proj* $\cap W \neq \{\}$ **using** ‹*intertwined S*› **unfolding** *intertwined-def*
using ‹$p \in$ *Q-proj*› ‹$p' \in$ *Q′-proj*› ‹*proj.quorum Q′-proj*› ‹*proj.quorum Q-proj*› *assms*(*5*) *assms*(*6*) *proj.quorum-of-def* **by** *auto*
show *?thesis*
using *Int-assoc* ‹*Q′-proj* $\subseteq Q' \cap W$› ‹*Q-proj* $\cap$ *Q′-proj* $\cap W \neq \{\}$› ‹*Q-proj* $\subseteq Q \cap W$› **by** *auto*
qed

definition *pseudo-blocked* **where**
pseudo-blocked R p $\equiv \forall\ Q$. *pseudo-quorum Q* $\wedge\ p \in Q \longrightarrow Q \cap R \neq \{\}$

lemma *pseudo-proj-is-intersection*:
assumes *pseudo-quorum q* **shows** *proj-of q* $= q \cap W$
using *assms* **unfolding** *pseudo-quorum-def proj-of-def* **apply** *auto*
using *inf.absorb-iff2* **by** *fastforce*

lemma *proj-of-pseudo-is-proj-quorum*:
assumes *pseudo-quorum q* **shows** *proj.quorum* (*proj-of q*)
using *assms* **unfolding** *proj.quorum-def pseudo-quorum-def proj-slices-def*
apply (*simp add*: *pseudo-proj-is-intersection*[*OF assms*(*1*)])
by (*meson Int-commute inf-le1 inf-le2 order-trans*)

lemma *l3′*: — needed?
assumes $S \subseteq Q \cap W$ **and** *pseudo-quorum Q*
shows $S \subseteq$ *proj-of Q*
using *assms* **unfolding** *pseudo-quorum-def proj-of-def*
using *contra-subsetD* **by** *fastforce*

lemma *pseudo-blocked-imp-quorum-blocking*:
pseudo-blocked R p $\Longrightarrow$ *quorum-blocking R p*
by (*simp add*: *pseudo-blocked-def pseudo-quorum-def quorum-def quorum-of-def slices.quorum-blocking-def*)

lemma *pseudo-blocked-imp-blocking*:*pseudo-blocked R p* $\Longrightarrow$ *blocking R p*
by (*simp add*: *pseudo-blocked-imp-quorum-blocking slices.quorum-blocking-blocking*)

definition *pseudo-blocking* **where**
pseudo-blocking R p $\equiv$ *blocking* $(R \cup (-W))$ *p*

~~lemma pseudo-blocked-imp-pseudo-blocking: pseudo-blocked R p ==> pseudo-blocking R p nitpick[card 'a=8, verbose, iter slices.blocking = 8] oops~~

end

5 Introducing ill-behaved participants

locale *well-behaved* = *projection*
— Now W is the set of well-behaved participants
begin

5.1 FX

definition *FX* **where** $FX \equiv \{p\ .\ \neg blocking\ (-W)\ p\}$

lemma $FX\text{-}in\text{-}W$:$FX \subseteq W$ **unfolding** *FX-def*
by (*metis Compl-iff blocking.intros*(*1*) *mem-Collect-eq subsetI*)

lemma *FX-has-quorum*:
assumes $p \in FX$ **obtains** Q **where** $p \in Q$ **and** *quorum* Q **and** $Q \subseteq FX$
by (*metis FX-def assms blocking.intros*(*2*) *mem-Collect-eq slices.quorum-def subsetI*)

lemma *FX-biggest*:
assumes $\bigwedge\ p\ .\ p \in FX' \Longrightarrow \exists\ Q\ .\ p \in Q \wedge quorum\ Q \wedge Q \subseteq W$
shows $FX' \subseteq FX$
using *assms* **by** (*force simp add*:*blocking-eq-quorum-blocking quorum-blocking-def quorum-of-def FX-def*)

5.2 The Intact set

interpretation *proj*: *slices proj-slices* **.**

definition *is-intact* **where**
$is\text{-}intact\ I \equiv I \subseteq W \wedge quorum\ I \wedge (\forall\ q_1\ q_2\ .$
$q_1 \cap I \neq \{\} \wedge q_2 \cap I \neq \{\} \wedge proj.quorum\ q_1 \wedge proj.quorum\ q_2 \longrightarrow q_1 \cap q_2 \neq \{\})$

lemma $intact\text{-}subseteq\text{-}W$:
assumes *is-intact* I **shows** $I \subseteq W$
using *assms is-intact-def* **by** *auto*

lemma *intact-subseteq-FX*:
assumes *is-intact* I **shows** $I \subseteq FX$
using *FX-biggest assms is-intact-def* **by** *auto*

~~lemma assumes quorum-blocking B p and p ∈ I and is-intact I shows proj.quorum-blocking B p nitpick[card 'a=3] oops~~

5.3 The properties needed for consensus

Note **lemma** *pseudo-quorum-intersection*

lemma *l1*:
  assumes *pseudo-blocked* B p **and** $p \in I$ **and** *is-intact* I
  shows $B \cap I \neq \{\}$
proof −
  obtain Q **where** *quorum-of* p Q **and** $Q \subseteq I$
    using *assms*(*2*,*3*) *is-intact-def quorum-of-def* **by** *auto*
  moreover
  have *pseudo-quorum* Q
    using ‹*quorum-of* p Q› *pseudo-quorum-def quorum-def quorum-of-def* **by** *auto*
  with ‹*pseudo-blocked* B p› **show** *?thesis* **unfolding** *pseudo-blocked-def*
    using *calculation*(*1*) *calculation*(*2*) *slices.quorum-of-def* **by** *fastforce*
qed

lemma *l2*:
  assumes $p \in I$ **and** *is-intact* I **and** *pseudo-quorum* Q **and** $p' \in Q \cap I$
  shows *quorum-blocking* (*proj-of* Q) p
proof −
  have *proj.quorum* (*proj-of* Q)
    by (*simp add*: *assms*(*3*) *proj-of-pseudo-is-proj-quorum*)
  moreover
  have *proj-of* $Q \cap I \neq \{\}$
    using *assms*(*2*−*4*) *inf-assoc is-intact-def pseudo-proj-is-intersection* **by** *auto*
  ultimately
  show *?thesis* **using** ‹$p \in I$› ‹*is-intact* I› **unfolding** *is-intact-def*
   by (*metis IntI emptyE proj.quorum-blocking-def proj.quorum-of-def proj-blocking-is-blocking*)

qed

lemma *l1'*: **assumes** $p \in I$ **and** *is-intact* I **and** *pseudo-blocking* R p **shows** $R \cap I \neq \{\}$
  — Note *pseudo-blocking ?R ?p* $\equiv$ *blocking* (*?R* $\cup$ − W) *?p*
proof −
  obtain Q **where** *quorum-of* p Q **and** $Q \subseteq I$
    using *assms*(*1*) *assms*(*2*) *is-intact-def quorum-of-def* **by** *auto*
  thus *?thesis* **using** ‹*pseudo-blocking* R p› *intact-subseteq-W* ‹*is-intact* I› *blocking-imp-quorum-blocking*
    unfolding *pseudo-blocking-def quorum-blocking-def*
    by *fastforce*
qed

5.4 union of intact is intact

interpretation *perso*:*personal* λ p . $\{q\ .\ p \in q \wedge proj.quorum\ q\}$ W
proof *standard*
  fix Q p
  assume $Q \in \{q.\ p \in q \wedge proj.quorum\ q\}$
  thus $p \in Q$ **by** *auto*

next
  fix $p\ p'\ Q$
  assume $Q \in \{q.\ p \in q \wedge proj.quorum\ q\}$ **and** $p' \in Q$
  thus $Q \in \{q.\ p' \in q \wedge proj.quorum\ q\}$
    using *proj.quorums-closed* **by** *fastforce*
qed

lemma *perso-intact-quorum-is-intact*:
  assumes *quorum I* **and** *perso.is-intact I* **shows** *is-intact I*
  using *assms* **unfolding** *is-intact-def perso.is-intact-def*
  by *blast*

lemma *proj-quorum-in-W*:
  assumes *proj.quorum Q* **and** $Q \cap I \neq \{\}$ **and** $I \subseteq W$
  obtains Q_w **where** $Q_w \subseteq W$ **and** $Q_w \subseteq Q$ **and** *proj.quorum* Q_w **and** $Q_w \cap I \neq \{\}$
proof −
 have *proj.quorum* $(Q \cap W)$ **using** *assms* **unfolding** *proj.quorum-def proj-slices-def*
    by(*auto*; (*metis inf-le2*))
  thus *?thesis*
  proof −
    have $I = I \cap W$
      by (*meson assms*(*3*) *inf.orderE*)
    then show *?thesis*
     by (*metis* (*no-types*) *Int-lower1 Int-lower2* ‹*proj.quorum* $(Q \cap W)$› *assms*(*2*) *inf.orderE inf-left-commute that*)
  qed
qed

lemma *stellar-intact-imp-perso-proj-intact*:
  assumes *is-intact I* **shows** *perso.is-intact I*
  unfolding *perso.is-intact-def*
proof (*intro conjI*)
  show $I \subseteq W$ **using** ‹*is-intact I*› **by** (*simp add*: *is-intact-def*)
next
  show $\forall p{\in}I.\ \exists Q{\in}\{q.\ p \in q \wedge proj.quorum\ q\}.\ Q \subseteq I$
    using ‹*is-intact I*› **using** *quorum-is-proj-quorum* **unfolding** *is-intact-def* **by** *auto*
next
  show $\forall p\ p'\ Q\ Q'.$
      $p \in I \wedge p' \in I \wedge Q \in \{q.\ p \in q \wedge proj.quorum\ q\} \wedge Q' \in \{q.\ p' \in q \wedge proj.quorum\ q\} \longrightarrow$
      $W \cap Q \cap Q' \neq \{\}$
  proof −
    have $W \cap Q \cap Q' \neq \{\}$
      if $p \in I$ **and** $p' \in I$ **and** $p \in Q$ **and** $p' \in Q'$ **and** *proj.quorum Q* **and** *proj.quorum Q'*
      for $p\ p'\ Q\ Q'$

proof −
have $I \subseteq W$
using *assms is-intact-def* **by** *blast*
have $Q \cap I \neq \{\}$ **and** $Q' \cap I \neq \{\}$
using *that*(*1*−*4*) **by** *blast*+
obtain Q_w **and** $Q_w{}'$ **where** $Q_w \cap I \neq \{\}$ **and** *proj.quorum* Q_w **and** $Q_w \subseteq Q$ **and** $Q_w \subseteq W$
and $Q_w{}' \cap I \neq \{\}$ **and** *proj.quorum* $Q_w{}'$ **and** $Q_w{}' \subseteq Q'$ **and** $Q_w{}' \subseteq W$
using *proj-quorum-in-W*
by (*metis* ‹$I \subseteq W$› ‹$Q \cap I \neq \{\}$› ‹$Q' \cap I \neq \{\}$› ‹*proj.quorum* Q'› ‹*proj.quorum* Q›)
have $Q_w \cap Q_w{}' \neq \{\}$ **using** ‹*is-intact* I› ‹$Q_w \cap I \neq \{\}$› ‹$Q_w{}' \cap I \neq \{\}$› ‹*proj.quorum* $Q_w{}'$› ‹*proj.quorum* Q_w›
unfolding *is-intact-def* **by** *blast*
thus *?thesis*
using ‹$Q_w \subseteq Q$› ‹$Q_w{}' \subseteq Q'$› ‹$Q_w{}' \subseteq W$› **by** *auto*
qed
thus *?thesis* **by** *blast*
qed
qed

lemma *intact-union*:
— Here we appeal to the union property proved in the personal model
assumes *is-intact* I_1 **and** *is-intact* I_2 **and** $I_1 \cap I_2 \neq \{\}$
shows *is-intact* $(I_1 \cup I_2)$
using *perso.intact-union assms is-intact-def perso-intact-quorum-is-intact quorum-union stellar-intact-imp-perso-proj-intact*
by *meson*

end

5.5 with quorum intersection

locale *quorum-intersection* = *well-behaved* +
assumes *quorum-intersection*:
$\bigwedge q_1\ q_2\ .\ [\![quorum\ q_1;\ quorum\ q_2]\!] \Longrightarrow q_1 \cap q_2 \neq \{\}$
begin

TODO: Anything to prove here?

end

6 elementary quorums

locale *elementary* = *slices*
begin

definition *elementary* **where**
$elementary\ s \equiv quorum\ s \wedge (\forall\ s'\ .\ s' \subset s \longrightarrow \neg quorum\ s')$

lemma *finite-subset-wf*:
  shows $wf\ \{(X,\ Y).\ X \subset Y \wedge finite\ Y\}$
  by (*metis finite-psubset-def wf-finite-psubset*)

lemma *quorum-contains-elementary*:
  assumes *finite s* **and** $\neg$ *elementary s* **and** *quorum s*
  shows $\exists\ s'\ .\ s' \subset s \wedge elementary\ s'$ **using** *assms*
proof (*induct s rule*:*wf-induct*[**where** $?r=\{(X,\ Y).\ X \subset Y \wedge finite\ Y\}$])
  case *1*
  then show *?case* **using** *finite-subset-wf* **by** *simp*
next
  case (*2 x*)
  then show *?case*
   by (*metis* (*full-types*) *elementary-def finite-psubset-def finite-subset in-finite-psubset less-le psubset-trans*)
qed

inductive *path* **where**
  path []
| $\bigwedge x\ .\ path\ [x]$
| $\bigwedge l\ n\ .\ [\![path\ l;\ S \in Q\ (hd\ l);\ n \in S]\!] \Longrightarrow path\ (n\#l)$

lemma *elementary-connected*:
  assumes *elementary s* **and** $n_1 \in s$ **and** $n_2 \in s$ **and** $n_1 \in W$ **and** $n_2 \in W$
  shows $\exists\ l\ .\ hd\ (rev\ l) = n_1 \wedge hd\ l = n_2 \wedge path\ l$ (**is** *?P*)
proof −
  { assume $\neg$*?P*
   define *x* **where** $x \equiv \{n \in s\ .\ \exists\ l\ .\ l \neq [] \wedge hd\ (rev\ l) = n_1 \wedge hd\ l = n \wedge path\ l\}$
    have $n_2 \notin x$ **using** ‹¬*?P*› *x-def* **by** *auto*
    have $n_1 \in x$ **unfolding** *x-def* **using** *assms*(*2*) *path.intros*(*2*) **by** *force*
    have *quorum x*
    proof −
      { fix *n*
        assume $n \in x$
        have $\exists\ S\ .\ S \in slices\ n \wedge S \subseteq x$
        proof −
          obtain *S* **where** $S \in slices\ n$ **and** $S \subseteq s$ **using** ‹*elementary s*› ‹$n \in x$› **unfolding** *x-def*
            by (*force simp add*:*elementary-def quorum-def*)
          have $S \subseteq x$
          proof −
            { assume $\neg\ S \subseteq x$
              obtain *m* **where** $m \in S$ **and** $m \notin x$ **using** ‹$\neg\ S \subseteq x$› **by** *auto*
              obtain l' **where** $hd\ (rev\ l') = n_1$ **and** $hd\ l' = n$ **and** $path\ l'$ **and** $l' \neq []$
                using ‹$n \in x$› *x-def* **by** *blast*
              have $path\ (m\ \#\ l')$ **using** ‹$path\ l'$› ‹$m \in S$› ‹$S \in slices\ n$› ‹$hd\ l' = n$›
                using *path.intros*(*3*) **by** *fastforce*

moreover have $hd\ (rev\ (m\ \#\ l'))\ =\ n_1$ **using** ‹$hd\ (rev\ l')\ =\ n_1$› ‹l'
$\neq$ []› **by** *auto*
ultimately have $m \in x$ **using** ‹$m \in S$› ‹$S \subseteq s$› *x-def* **by** *auto*
hence *False* **using** ‹$m \notin x$› **by** *blast* **}**
thus *?thesis* **by** *blast*
qed
thus *?thesis*
using ‹$S \in$ *slices* n› **by** *blast*
qed }
thus *?thesis* **by** (*metis quorum-def*)
qed
moreover have $x \subset s$
using ‹$n_2 \notin x$› *assms*(*3*) *x-def* **by** *blast*
ultimately have *False* **using** ‹*elementary s*›
using *elementary-def* **by** *auto*
}
thus *?P* **by** *blast*
qed

end

7 Bracha Broadcast

record ($'p$, $'val$) *state* =
voted :: $'p \Rightarrow\ 'val \Rightarrow$ *bool*
accepted :: $'p \Rightarrow\ 'val \Rightarrow$ *bool*
committed :: $'p \Rightarrow\ 'val \Rightarrow$ *bool*

locale *bracha* = *well-behaved*
begin

definition *vote* **where**
vote $s\ s'\ p\ v \equiv$
$(\forall\ v\ .\ \neg$ *voted* $s\ p\ v)$
$\wedge\ s' = s(|$*voted* := (*voted* s)(p := (*voted* $s\ p$)(v := *True*))$|)$

definition *accept* **where**
accept $s\ s'\ p\ v \equiv$
($(\exists\ Q\ .$ *pseudo-quorum* $Q \wedge p \in Q \wedge (\forall\ p' \in Q\ .$ *voted* $s\ p'\ v))$
$\vee\ (\exists\ B\ .$ *pseudo-blocking* $B\ p \wedge (\forall\ p' \in B\ .$ *accepted* $s\ p'\ v)))$
$\wedge\ s' = s(|$*accepted* := (*accepted* s)(p := (*accepted* $s\ p$)(v := *True*))$|)$

definition *commit* **where**
commit $s\ s'\ p\ v \equiv$
$(\exists\ Q\ .$ *pseudo-quorum* $Q \wedge p \in Q \wedge (\forall\ p' \in Q\ .$ *accepted* $s\ p'\ v))$
$\wedge\ s' = s(|$*committed* := (*committed* s)(p := (*committed* $s\ p$)(v := *True*))$|)$

definition *trans* **where**
trans $s\ s' \equiv \exists\ v\ p\ .$

vote s s′ p v ∨ *accept s s′ p v* ∨ *commit s s′ p v*

end

7.1 Safety proof

declare *if-splits*[*split*]

method *rw-record-expr* **for** s =
  (*cases s*; *simp*; *match* **premises in** *P*[*thin*]:*s* = - ⇒ ‹−›)

locale *intact* = *bracha* +
  fixes *I*
  assumes *I-intact*:*is-intact I*
begin

interpretation *proj*: *slices proj-slices* **.**

definition *is-inductive* **where**
  is-inductive i ≡ ∀ *s s′* . *i s* ∧ *trans s s′* ⟶ *i s′*

definition *invariant-1* **where**
  invariant-1 s ≡ ∀ *p v w* . *p* ∈ *W* ∧ *voted s p v* ∧ *voted s p w* ⟶ *v* = *w*

definition *invariant-2* :: (′*a*,′*b*)*state* ⇒ *bool* **where**
  invariant-2 s ≡ ∀ *p v* . *p* ∈ *I* ∧ *accepted s p v*
    ⟶ (∃ *Q* . *proj.quorum Q* ∧ *Q* ∩ *I* ≠ {} ∧ (∀ *p′* ∈ *Q* . *voted s p′ v*))

lemma *invariant-1*:
  is-inductive invariant-1
  unfolding *is-inductive-def invariant-1-def trans-def vote-def accept-def commit-def*
  by (*auto*)

declare *if-splits*[*split*]
declare *quorum-of-def*[*simp*]

lemma *invariant-2*:
  is-inductive invariant-2
proof −
  {
    fix *s s′* :: (′*a*,′*b*)*state*
    assume *invariant-2 s* **and** *trans s s′*
    have *invariant-2 s′*
    proof −
      have *invariant-2 s′* **if** *invariant-2 s* **and** *vote s s′ p v* **for** *p v*
        using *that* **unfolding** *invariant-2-def vote-def*
        by (*cases s*; *auto*; *cases s′*; *auto*; *metis*)
      moreover
      have *invariant-2 s′* **if** *invariant-2 s* **and** *commit s s′ p v* **for** *p v*

using *that* **unfolding** *invariant-2-def commit-def*
by (*cases s*; *auto*; *cases s'*; *auto*; *metis*)
moreover
have *invariant-2 s'* **if** *invariant-2 s* **and** *accept s s' p v* **for** *p v*
using *that I-intact* **unfolding** *invariant-2-def accept-def*
apply (*cases s*; *auto*; *cases s'*; *auto*)
apply (*smt IntE IntI emptyE intact-subseteq-W proj-of-pseudo-is-proj-quorum pseudo-proj-is-intersection set-rev-mp*)
apply (*metis Compl-iff disjoint-eq-subset-Compl subsetI well-behaved.l1'*)+
done
ultimately
show *?thesis* **by** (*meson* ‹*invariant-2 s*› ‹*local.trans s s'*› *local.trans-def*)
qed }
thus *?thesis* **unfolding** *is-inductive-def* **by** *auto*
qed

To continue...

end

~~*section Personal elementarytext this has no use so farlocale personal-elementary = slicesbegindefinition elementary where elementary p Q ≡ quorum-of p Q ∧ (∀ Q'. ¬(quorum-of p Q' ∧ Q' ⊂ Q))lemma elementary p Q ⟹ ∄! S ∈ slices p . S ⊈ Q nitpick[verbose, card 'a=3] oopslemma assumes elementary p Q obtains S f where S ∈ slices p and Q = insert p (⋃ q∈S . f q) and ⋀ q . q ∈ S ⟹ quorum-of q (f q) oopsend*~~

end