Safety Proof of a Distributed Termination-Detection Algorithm

Giuliano Losa

December 4, 2022

Contents

theory *Termination*
  imports *Main HOL−Statespace.StateSpaceSyntax*
begin

1 Specification of the algorithm

statespace $'p$ *vars* =
  $s :: 'p \Rightarrow 'p \Rightarrow nat$
  $S :: 'p \Rightarrow 'p \Rightarrow nat$
  $r :: 'p \Rightarrow 'p \Rightarrow nat$
  $R :: 'p \Rightarrow 'p \Rightarrow nat$
  $visited :: 'p \Rightarrow bool$
  $terminated :: bool$

context *vars*
begin

definition *pending* **where**
  — Number of messages in flight from p to q
  $pending\ c\ p\ q \equiv ((c{\cdot}s)\ p\ q) - ((c{\cdot}r)\ q\ p)$

definition *receive-step* **where**
  — Process p receives a message from q and sends a few more messages in response.
  $receive\text{-}step\ c\ c'\ p \equiv \exists\ q\ .$
    $p \neq q$
    $\wedge\ pending\ c\ q\ p > 0$
    $\wedge\ (\exists\ P\ .$ — We pick a set P of processes to send messages to.
      $\exists\ s\text{-}p'\ .\ s\text{-}p' = (\lambda\ q\ .$
        $let\ s\text{-}p\text{-}q = ((c{\cdot}s)\ p\ q)$
        $in\ if\ q \in P\ then\ s\text{-}p\text{-}q + 1\ else\ s\text{-}p\text{-}q)$

```
— The new state:
  ∧ c′ = c<s := (c·s)(p:=s-p′), r := (c·r)(p:=((c·r) p)(q := (c·r) p q + 1))>)

definition daemon-step where
  daemon-step c c′ p ≡ ¬ c·terminated ∧ (
    if (∃ p . ¬ (c·visited) p) ∨ (∃ p q . (c·S) p q ≠ (c·R) q p)
    then c′ = c<
      visited := (λ q . (c·visited) q ∨ p = q),
      S := (c·S)(p := (c·s) p),
      R := (c·R)(p := (c·r) p)>
    else c′ = c<terminated := True> )

definition step where
  step c c′ ≡ ∃ p . receive-step c c′ p ∨ daemon-step c c′ p

definition init where
  init c ≡ ∀ p q .
    (c·s) p q ≥ 0
    ∧ (c·S) p q = 0
    ∧ (c·r) p q = 0
    ∧ (c·R) p q = 0
    ∧ ¬ (c·visited) p
    ∧ ¬ (c·terminated)
```

2 Correctness proof

```
definition inv1 where
  — A process can only receive what has been sent.
  inv1 c ≡ ∀ p q . (c·r) p q ≤ (c·s) q p

lemma inv1-init:
  assumes init c
  shows inv1 c
  using assms
  unfolding init-def inv1-def
  by auto

lemma inv1-step:
  assumes step c c′ and inv1 c
  shows inv1 c′
proof –
  have inv1 c′ if receive-step c c′ p  and inv1 c for p
    using that unfolding receive-step-def pending-def inv1-def
    by (auto; smt (verit, best) trans-le-add1)
  moreover
  have inv1 c′ if daemon-step c c′ p and inv1 c for p
    using that unfolding daemon-step-def inv1-def
    by (auto split:if-splits)
  ultimately show ?thesis
```

using *assms step-def* **by** *blast*
qed

definition *consistent* **where**
$consistent\ c\ Q \equiv \forall\ p \in Q\ .\ (c{\cdot}visited)\ p \wedge (\forall\ q \in Q\ .\ (c{\cdot}S)\ p\ q = (c{\cdot}R)\ q\ p)$

definition *inv2* **where**
$inv2\ c \equiv \forall\ Q\ .\ consistent\ c\ Q \wedge (\exists\ p \in Q\ .\ \exists\ q\ .\ (c{\cdot}R)\ p\ q \neq (c{\cdot}r)\ p\ q \vee (c{\cdot}S)\ p\ q \neq (c{\cdot}s)\ p\ q)$
$\longrightarrow (\exists\ p \in Q\ .\ \exists\ q \in -Q\ .\ (c{\cdot}r)\ p\ q > (c{\cdot}R)\ p\ q)$

lemma *inv2-init*:
assumes *init c*
shows *inv2 c*
using *assms*
unfolding *init-def inv2-def consistent-def*
by *auto*

lemma *inv2-step*:
assumes *step c c′* **and** *inv1 c* **and** *inv1 c′* **and** *inv2 c*
shows *inv2 c′*
proof −
define *stale* **where** $stale\ c\ Q \equiv \exists\ p \in Q\ .\ \exists\ q\ .\ (c{\cdot}R)\ p\ q \neq (c{\cdot}r)\ p\ q \vee (c{\cdot}S)\ p\ q \neq (c{\cdot}s)\ p\ q$ **for** *c Q*
have *inv2 c′* **if** *receive-step c c′ p* **for** *p*
proof −
{ fix *Q*
assume *consistent c′ Q* **and** *stale c′ Q*
have $\exists\ p \in Q\ .\ \exists\ q \in -Q\ .\ (c'{\cdot}r)\ p\ q > (c'{\cdot}R)\ p\ q$
proof −
have *consistent c Q* **using** ‹*consistent c′ Q*› **and** ‹*receive-step c c′ p*›
unfolding *consistent-def receive-step-def* **by** *auto*
{ assume *stale c Q*
— If Q is stale in *c*, then already in *c* there is a process that has received a message from outside Q that the daemon has not seen. This remains true.
hence $\exists\ p \in Q\ .\ \exists\ q \in -Q\ .\ (c{\cdot}r)\ p\ q > (c{\cdot}R)\ p\ q$ **using** ‹*inv2 c*› ‹*consistent c Q*› *inv2-def stale-def* **by** *auto*
hence *?thesis* **using** ‹*receive-step c c′ p*› **unfolding** *receive-step-def*
apply *auto*
apply (*metis* (*mono-tags*, *opaque-lifting*) *ComplI less-Suc-eq*)
done }
moreover
{ assume ¬ (*stale c Q*)
— If Q is not stale in *c*, then no process in Q can receive a message from another process in Q (because all counts match). So, because we assume that the count of at least one process in Q changes, it must be by receiving a message from outside Q.
obtain *q* **where** $p \in Q$ **and** $(c'{\cdot}r)\ p\ q \neq (c{\cdot}r)\ p\ q$
using ‹*stale c′ Q*› **and** ‹*receive-step c c′ p*› **and** ‹¬ (*stale c Q*)›

unfolding *receive-step-def stale-def pending-def*
apply *auto*
apply (*smt* (*verit*, *best*) *n-not-Suc-n*)+
done
moreover
have $q \notin Q$
proof –
have $\forall\ p \in Q\ .\ \forall\ q \in Q\ .\ (c{\cdot}r)\ p\ q = (c'{\cdot}r)\ p\ q$
proof –
from ‹$\neg$ (*stale c Q*)› **have** $\forall\ p \in Q\ .\ \forall\ q \in Q\ .\ (c{\cdot}s)\ p\ q = (c{\cdot}r)\ q\ p$
using ‹*consistent c Q*› *consistent-def stale-def* **by** *force*
thus *?thesis*
using ‹*receive-step c c′ p*› *pending-def* **unfolding** *receive-step-def* **by** *auto*
qed
thus *?thesis*
using ‹$(c'{\cdot}r)\ p\ q \neq (c{\cdot}r)\ p\ q$› ‹$p \in Q$› **by** *auto*
qed
moreover
have $(c'{\cdot}r)\ p\ q > (c{\cdot}r)\ p\ q$ **using** ‹$(c'{\cdot}r)\ p\ q \neq (c{\cdot}r)\ p\ q$› **and** ‹*receive-step c c′ p*›
unfolding *receive-step-def* **by** (*auto split*:*if-splits*)
moreover
have $(c'{\cdot}R)\ p\ q = (c{\cdot}r)\ p\ q$ **using** ‹*receive-step c c′ p*› **and** ‹$\neg$ (*stale c Q*)› **and** ‹$p \in Q$›
unfolding *receive-step-def stale-def* **by** *auto*
ultimately
have *?thesis* **by** *force* **}**
ultimately show *?thesis* **by** *auto*
qed }
thus *?thesis* **unfolding** *inv2-def stale-def* **by** *blast*
qed
moreover
have *inv2 c′* **if** *daemon-step c c′ p* **for** *p*
proof –
{ fix *Q*
assume *consistent c′ Q* **and** *stale c′ Q*
have $\exists\ p \in Q\ .\ \exists\ q \in -Q\ .\ (c'{\cdot}r)\ p\ q > (c'{\cdot}R)\ p\ q$
proof (*cases* $(\exists\ p\ .\ \neg\ (c{\cdot}visited)\ p) \vee (\exists\ p\ q\ .\ (c{\cdot}S)\ p\ q \neq (c{\cdot}R)\ q\ p)$)
— Here we do a case analysis of the condition in the if branch of the daemon step.
case *True*
then show *?thesis*
proof –
{ assume $p \notin Q$ — The daemon visits a process not in Q
have $\exists\ p \in Q\ .\ \exists\ q \in -Q\ .\ (c{\cdot}r)\ p\ q > (c{\cdot}R)\ p\ q$
proof –
from ‹$p \notin Q$› **have** *consistent c Q* **and** *stale c Q*
using ‹*daemon-step c c′ p*› **and** ‹*consistent c′ Q*›

and ‹*stale* c' Q›
unfolding *daemon-step-def consistent-def stale-def*
by (*force split*: *if-splits*)+
thus *?thesis* **using** ‹*inv2* c› **unfolding** *inv2-def stale-def* **by** *auto*
qed
hence *?thesis* **using** ‹*daemon-step* c c' p› **and** ‹$p \notin Q$›
unfolding *daemon-step-def* **by** (*auto split*: *if-splits*) **}**
moreover
{ assume $p \in Q$ — The daemon visits a process in Q
define Q' **where** $Q' \equiv Q - \{p\}$

First we show that Q' is consistent but stale. So, by *inv2* c, the daemon missed a message from outside Q'. Then it remains to show that this message did not come from p

obtain p' q **where** $p' \in Q'$ **and** $q \in -Q'$ **and** $(c'\cdot r)\ p'\ q > (c'\cdot R)\ p'\ q$
proof −
have $\exists\ p \in Q'\ .\ \exists\ q \in -Q'\ .\ (c'\cdot r)\ p\ q > (c'\cdot R)\ p\ q$
proof −
have $\exists\ p \in Q'\ .\ \exists\ q \in -Q'\ .\ (c\cdot r)\ p\ q > (c\cdot R)\ p\ q$
proof −
have *consistent* c Q'
using ‹*daemon-step* c c' p› *True* ‹*consistent* c' Q›
unfolding *daemon-step-def consistent-def* Q'*-def*
by (*auto*; (*smt* (*verit*)))
moreover
have *stale* c Q'
using ‹*daemon-step* c c' p› *True* ‹*stale* c' Q›
unfolding *daemon-step-def* Q'*-def stale-def*
by (*auto split*: *if-splits*)
ultimately
show *?thesis*
using ‹*inv2* c› **unfolding** *inv2-def stale-def* **by** *auto*
qed
thus *?thesis* **using** ‹*daemon-step* c c' p› **unfolding** *daemon-step-def* Q'*-def*
by (*auto split*: *if-splits*)
qed
thus *?thesis* **using** *that* **by** *auto*
qed
moreover
have $q \neq p$
— Then it remains to show that the message that the daemon missed did not come from p.
proof −
have $(c'\cdot R)\ p'\ p = (c'\cdot s)\ p\ p'$
proof −
have $(c'\cdot r)\ p = (c'\cdot R)\ p$ **and** $(c'\cdot s)\ p = (c'\cdot S)\ p$
using ‹*daemon-step* c c' p› *True* **unfolding** *daemon-step-def*
by *auto*

moreover
have $(c'{\cdot}R)\ p'\ p = (c'{\cdot}S)\ p\ p'$ **using** ‹*consistent* c' Q› ‹$p \in Q$› ‹$p' \in Q'$›
unfolding *consistent-def* Q'*-def*
by *auto*
ultimately show *?thesis* **by** *auto*
qed
{ assume $p = q$
hence $(c'{\cdot}r)\ p'\ p > (c'{\cdot}R)\ p'\ p$ **using** ‹$(c'{\cdot}r)\ p'\ q > (c'{\cdot}R)\ p'\ q$›
by *auto*
hence $(c'{\cdot}r)\ p'\ p > (c'{\cdot}s)\ p\ p'$ **using** ‹$(c'{\cdot}R)\ p'\ p = (c'{\cdot}s)\ p\ p'$›
by *auto*
hence *False* **using** ‹*inv1* c'› **unfolding** *inv1-def*
by (*simp add*: *leD*) **}**
thus *?thesis* **by** *blast*
qed
ultimately
have *?thesis* **using** Q'*-def* **by** *blast*
}
ultimately show *?thesis* **by** *blast*
qed
next
case *False*
— Case in which the daemon declares termination; trivial because $c{\cdot}$*terminated* is the only thing that changes
then show *?thesis*
using ‹*daemon-step* c c' p› **and** ‹*consistent* c' Q›
and ‹*stale* c' Q›
and ‹*inv2* c›
unfolding *daemon-step-def consistent-def inv2-def stale-def*
by *auto*
qed }
thus *?thesis*
using *inv2-def stale-def* **by** *blast*
qed
ultimately show *?thesis*
using *assms*(*1*) **unfolding** *step-def* **by** *blast*
qed

definition *inv3* **where**
inv3 $c \equiv c{\cdot}$*terminated* $\longrightarrow$ *consistent* c *UNIV*

lemma *inv3-init*:
assumes *init* c
shows *inv3* c
using *assms*
unfolding *init-def inv3-def*
by *auto*

lemma *inv3-step*:
  assumes *step c c'* **and** *inv3 c*
  shows *inv3 c'*
  using *assms*
  unfolding *step-def daemon-step-def receive-step-def inv3-def consistent-def*
  by (*force split*:*if-splits*)

definition *safety* **where**
  safety c ≡ *c*·*terminated* ⟶ (∀ *p q* . *pending c p q* = *0*)

lemma *safe*:
  assumes *inv2 c* **and** *inv3 c*
  shows *safety c*
  using *assms* **unfolding** *inv2-def safety-def pending-def inv3-def consistent-def*
  by (*simp*; *metis ComplD iso-tuple-UNIV-I le-refl*)

end

end