Formal verification of a classic distributed algorithm using inductive invariants

A proof pearl

Giuliano Losa
Stellar Development Foundation

Plan for the talk

1. A cute distributed-computing problem and its solution

   Kumar, D. *A class of termination detection algorithms for distributed computations*, 1985

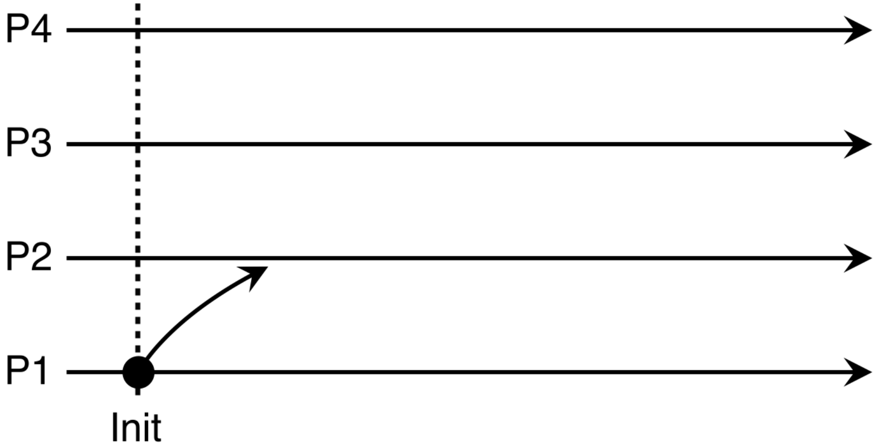
   Also in: Chandy and Misra. "Proofs of distributed algorithms: An exercise.", 1990

2. A visual argument of correctness
3. A new proof using an inductive invariant
   - Bounded verification with TLA+ and Apalache
   - Mechanically-checked proof in Isabelle/HOL
4. Fun exercises left to the audience

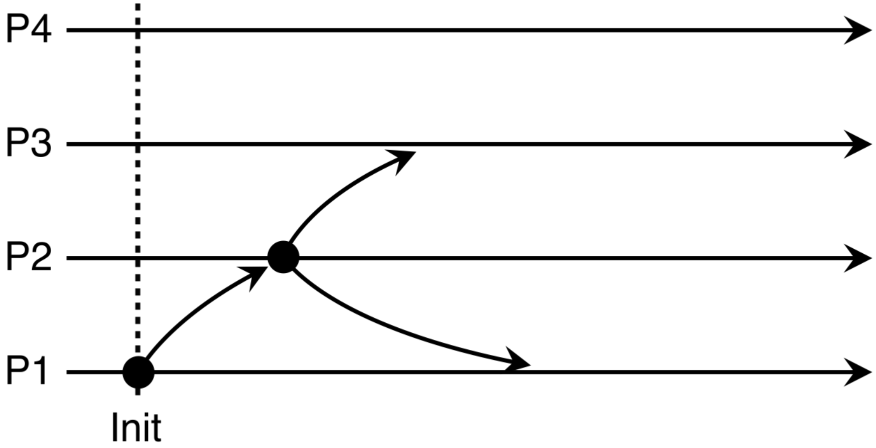
Model: message-driven, asynchronous distributed computation

- A set P of N asynchronous processes
- Asynchronous network
- Directed channel <p,q> between every pair of processes
- Every message sent is eventually delivered, and only sent messages are delivered
- No process failures
- Initially, some messages are in flight
- Upon receiving a message, a process sends zero or more messages
- A process only sends messages in upon receive a message

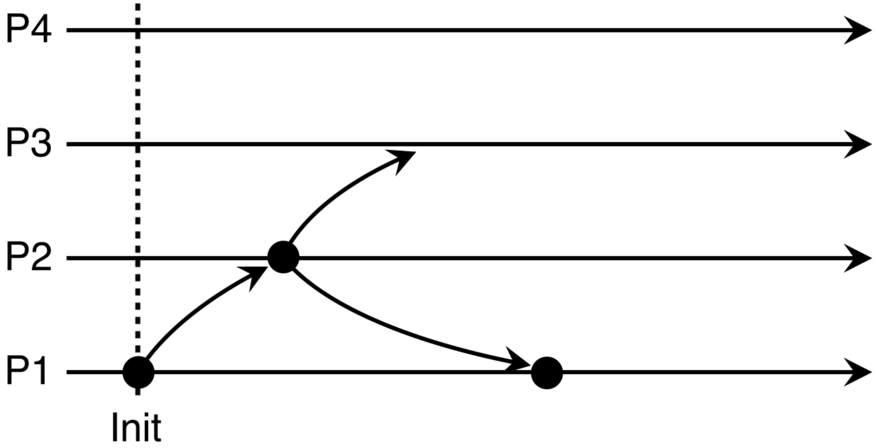
The computation terminates once there are no more messages in flight



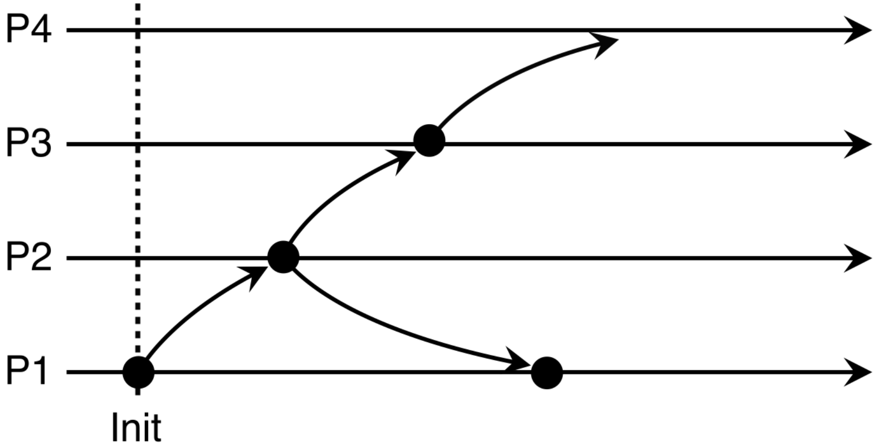
The computation terminates once there are no more messages in flight



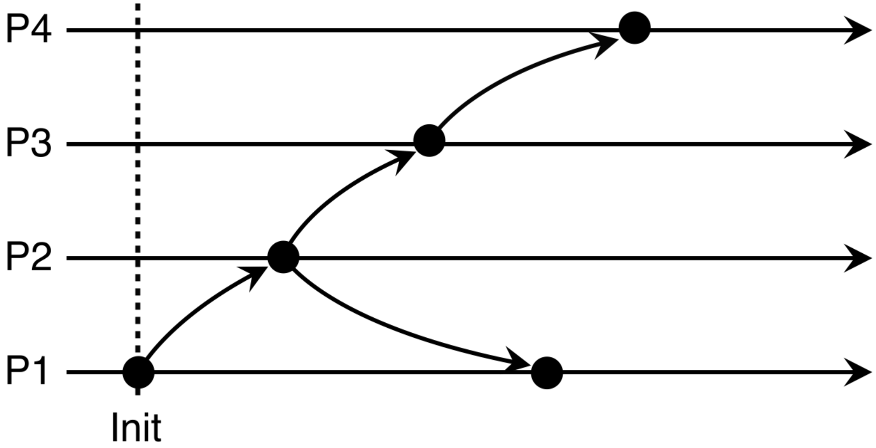
The computation terminates once there are no more messages in flight



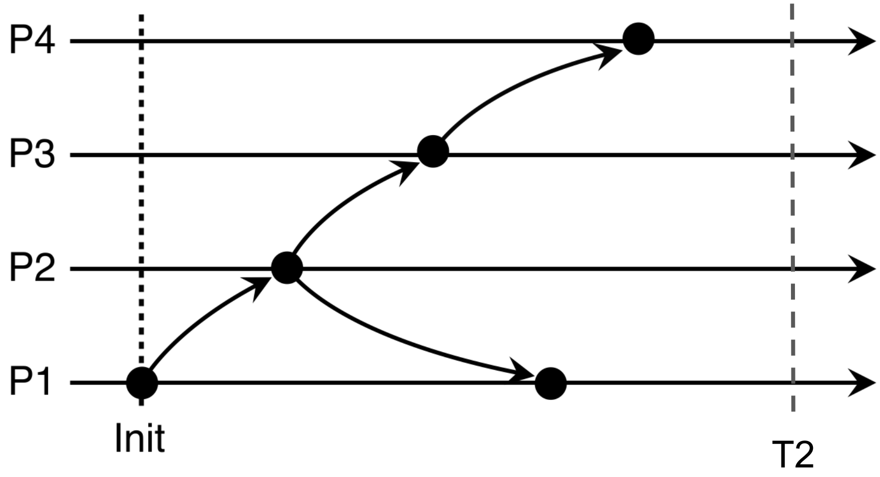
The computation terminates once there are no more messages in flight



The computation terminates once there are no more messages in flight



The computation terminates once there are no more messages in flight

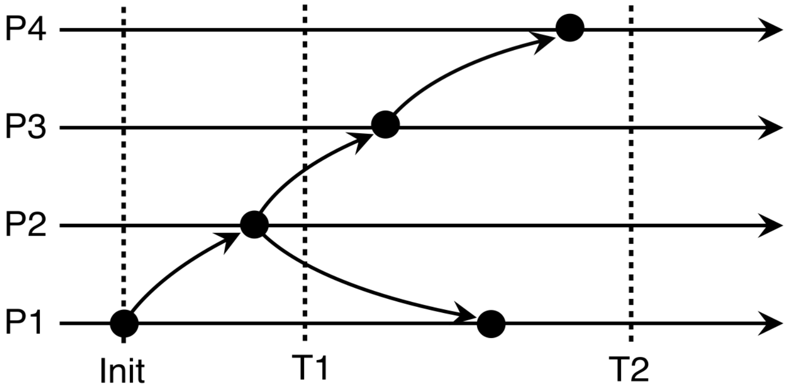


At time T2, there are no more messages in flight and the computation has terminated

Problem: detect termination

- Additional process: the Daemon
- If the system terminates, then eventually, the daemon must declare that the system has terminated
- When the daemon declares termination, the system must indeed have terminated

Processes count messages on each channel



At time T1, s and r are inconsistent

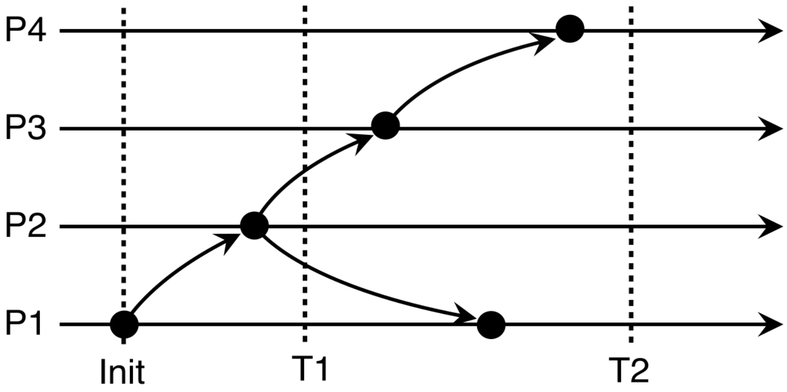
$s_{T1} =$

	P1	P2	P3	P4
P1		1		
P2	1			
P3		1		
P4				

$r_{T1} =$

	P1	P2	P3	P4
P1		0		
P2	1			
P3		0		
P4				

Processes count messages on each channel



At time T2, send and receive counts are consistent

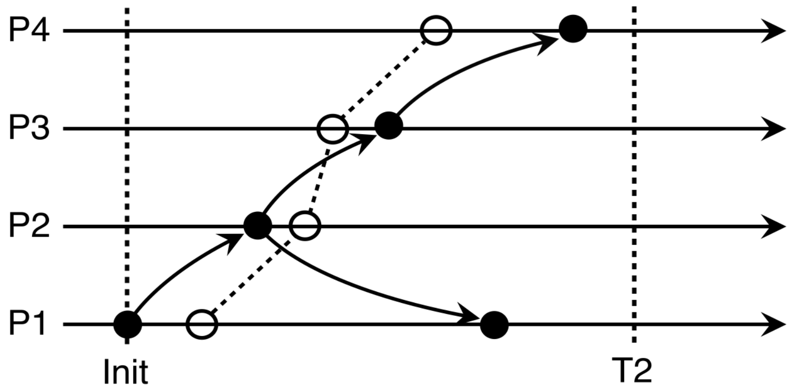
$s_{T1} =$

	P1	P2	P3	P4
P1		1		
P2	1			
P3		1		
P4			1	

$r_{T1} =$

	P1	P2	P3	P4
P1		1		
P2	1			
P3		1		
P4			1	

The daemon asks processes for their counts, remembers them, and uses that information to detect termination



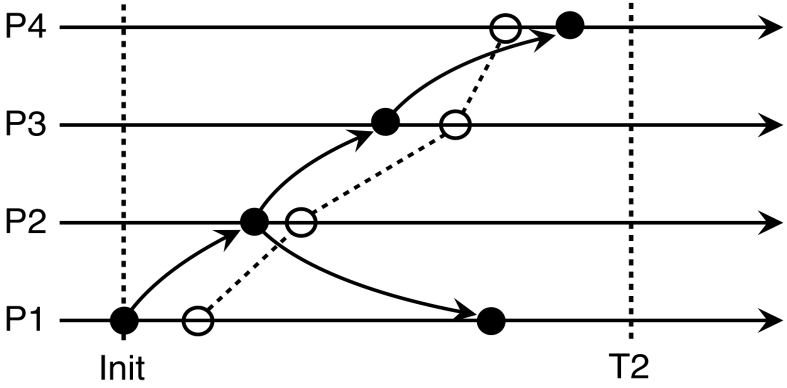
$s_{T1} =$

	P1	P2	P3	P4
P1		1		
P2	1			
P3		1		
P4			0	

$r_{T1} =$

	P1	P2	P3	P4
P1		0		
P2	1			
P3		0		
P4			0	

Another example



Daemon counts are inconsistent

$s_{T1} =$

	P1	P2	P3	P4
P1		1		
P2	1			
P3		1		
P4			1	

$r_{T1} =$

	P1	P2	P3	P4
P1		0		
P2	1			
P3		1		
P4			0	

When can the daemon declare termination?

Idea: take atomic snapshot (double-collect) and declare termination when the snapshot is consistent (number of messages sent on a channel equals to the number received)

Simpler: Declare termination as soon as the numbers collected so far are consistent

$\text{CONSTANT } P$ the set of Processors

$$\text{VARIABLES} \quad s,\ r,\ ds,\ dr,\ terminated$$

$$\begin{aligned}
&Init \triangleq \\
&\quad \land s \in [P \times P \to Nat] \\
&\quad \land \exists p, q \in P : s[\langle p, q\rangle] > 0 \quad \text{at least one message in flight} \\
&\quad \land r = [c \in P \times P \mapsto 0] \\
&\quad \land ds = [c \in P \times P \mapsto 0] \\
&\quad \land dr = [c \in P \times P \mapsto 0] \\
&\quad \land terminated = \text{FALSE}
\end{aligned}$$

$$\begin{aligned}
&NumPending(p, q) \triangleq \\
&\quad s[\langle p, q\rangle] - r[\langle p, q\rangle]
\end{aligned}$$

$$\begin{aligned}
&Process(self) \triangleq \\
&\quad \land \exists p \in P \setminus \{self\} : \quad \text{receive a message from } p \\
&\qquad \land NumPending(p, self) > 0 \\
&\qquad \land r' = [r \text{ EXCEPT } ![\langle p, self\rangle] = @ + 1] \\
&\quad \land \exists Q \in \text{SUBSET } (P \setminus \{self\}) : \quad \text{send a message to each member of set } Q \\
&\qquad \land s' = [c \in P \times P \mapsto \\
&\qquad\qquad \text{IF } c[1] = self \land c[2] \in Q \text{ THEN } s[c] + 1 \text{ ELSE } s[c]] \\
&\quad \land \text{UNCHANGED } \langle ds, dr, terminated\rangle
\end{aligned}$$

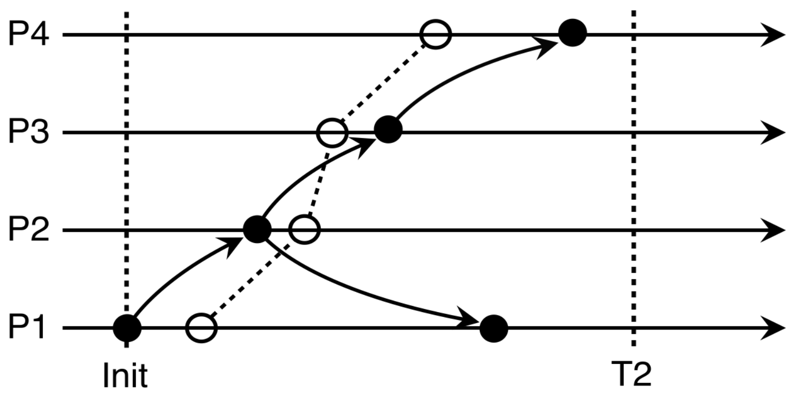
$$\begin{aligned}
&Consistent \triangleq \\
&\quad \land \forall p, q \in P : ds[\langle p, q\rangle] = dr[\langle p, q\rangle]
\end{aligned}$$

$$\begin{aligned}
&Daemon \triangleq \\
&\quad \land \text{IF } \neg Consistent \lor \forall p, q \in P : ds[\langle p, q\rangle] = 0 \\
&\qquad \text{THEN } \exists p \in P : \quad \text{pick a process } p \text{ to visit} \\
&\qquad\quad \land \quad ds' = [c \in P \times P \mapsto \text{IF } c[1] = p \text{ THEN } s[c] \text{ ELSE } ds[c]] \\
&\qquad\quad \land \quad dr' = [c \in P \times P \mapsto \text{IF } c[2] = p \text{ THEN } r[c] \text{ ELSE } dr[c]] \\
&\qquad\quad \land \quad \text{UNCHANGED } terminated \\
&\qquad \text{ELSE} \\
&\qquad\quad \land terminated' = \text{TRUE} \quad \text{declare termination} \\
&\qquad\quad \land \text{UNCHANGED } \langle ds, dr\rangle \\
&\quad \land \text{UNCHANGED } \langle s, r\rangle
\end{aligned}$$

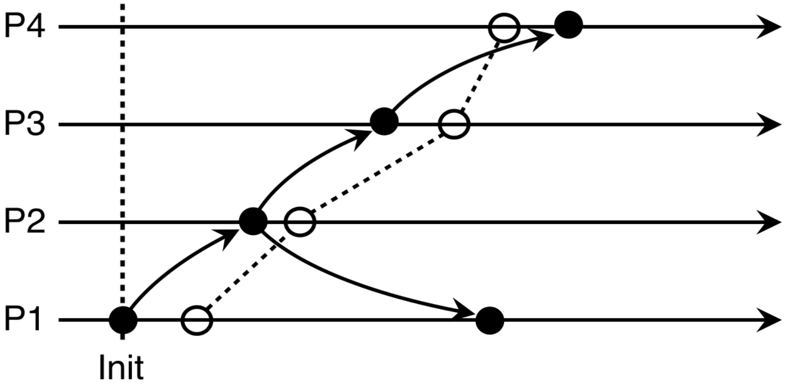
$$Next \triangleq Daemon \lor (\exists p \in P : Process(p))$$

$$Safety \triangleq terminated \Rightarrow \forall p, q \in P : NumPending(p, q) = 0$$

The set of last visits defines *the wave*



- Activity to the left of the wave is recorded
- Activity to the right or the wave is not recorded
- Messages that cross cause discrepancies in sent vs received count



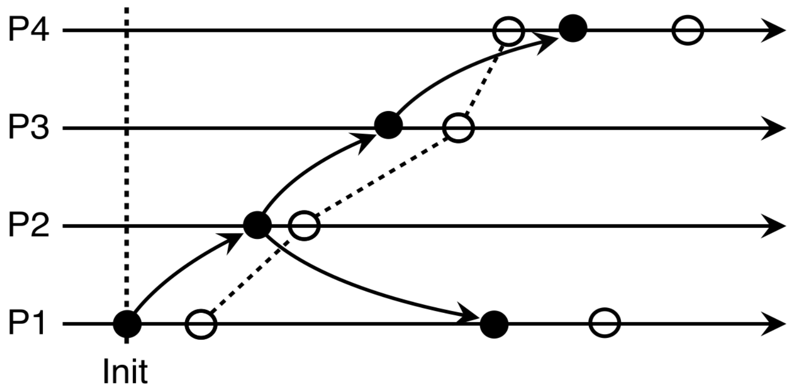
Deamon does not detect termination

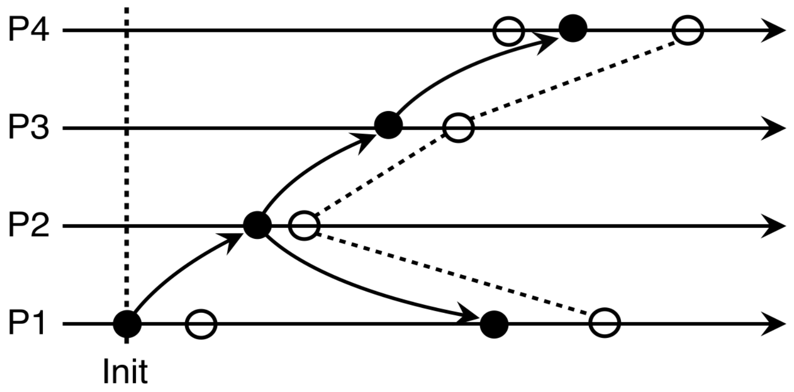
$ds_{T2} =$

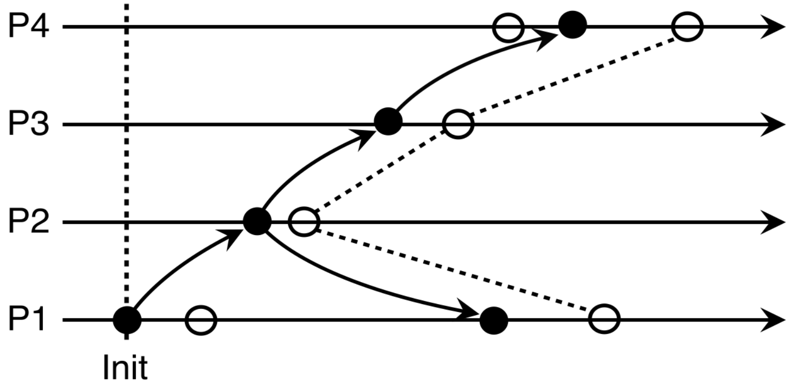
	P1	P2	P3	P4
P1		1		
P2	1			
P3		1		
P4			1	

$dr_{T2} =$

	P1	P2	P3	P4
P1		0		
P2	1			
P3		1		
P4			0	


P4
P3
P2
P1
Init


P4
P3
P2
P1
Init



Deamon detects termination

$ds_{T2} =$

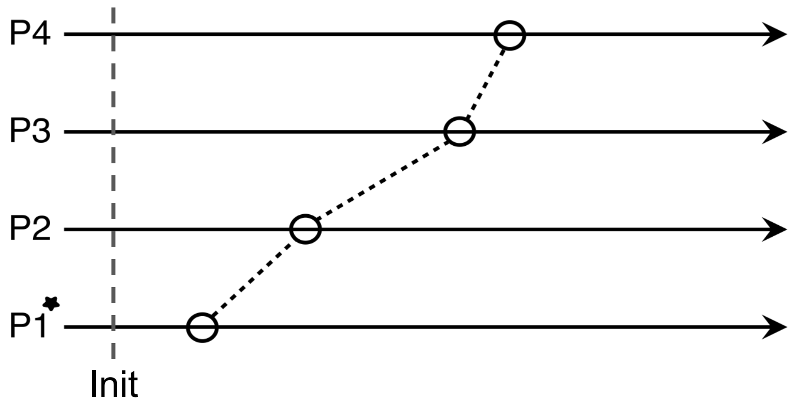
	P1	P2	P3	P4
P1		1		
P2	1			
P3		1		
P4			1	

$dr_{T2} =$

	P1	P2	P3	P4
P1		1		
P2	1			
P3		1		
P4			1	

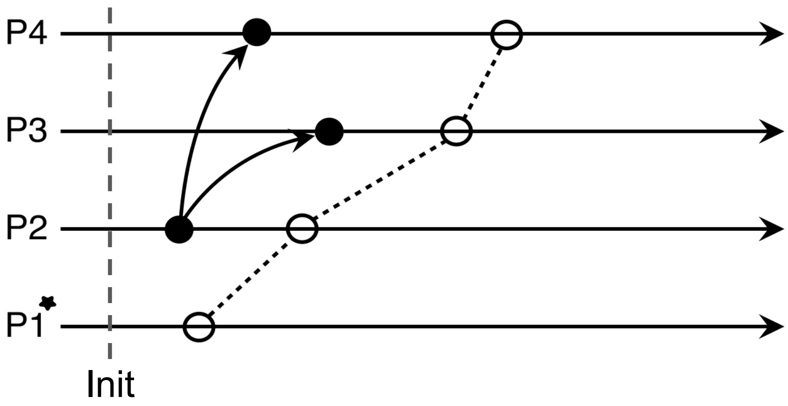
Visual demonstration of correctness

Assume all daemon counts match at the end



Visual demonstration of correctness

Assume all daemon counts match at the end

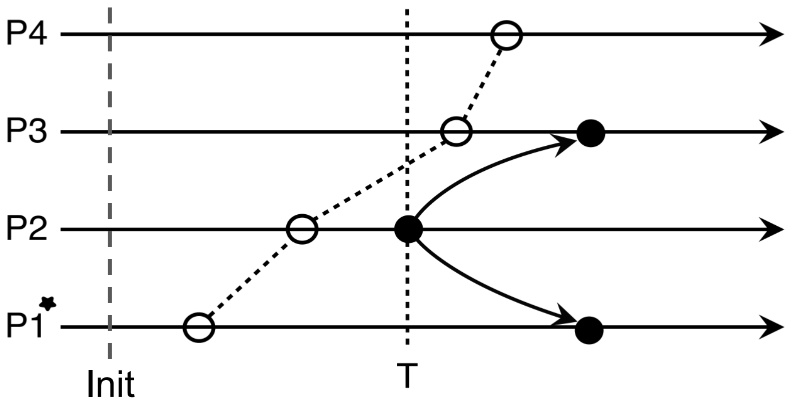


Case 1: there is activity only on the left of the wave

Then the daemon's counts are the real counts and thus no message can be pending

Visual demonstration of correctness

Assume all daemon counts match at the end



Case 2: assume there is activity to the right of the wave

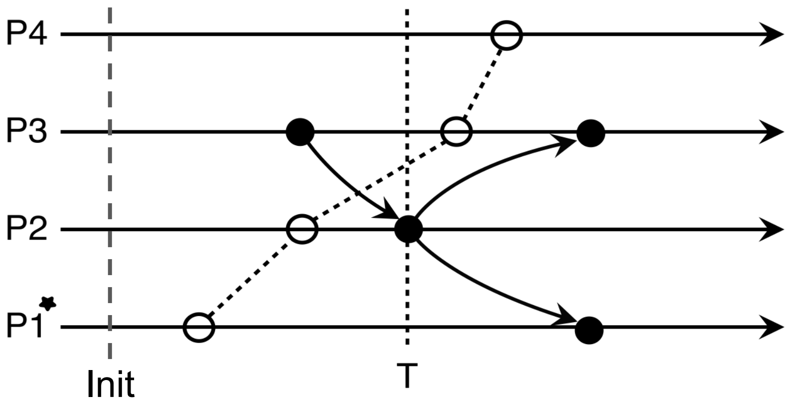
Let p be the process that sends the earliest message to the right of the wave (p = P2)

Note T > Init

So P received a message from the left of the wave, or otherwise p would not be the earliest to send on the right

Visual demonstration of correctness

Assume all daemon counts match at the end

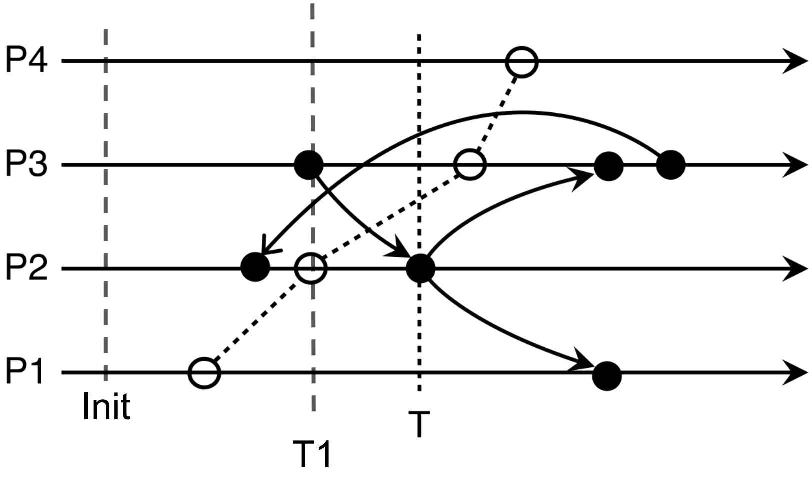


P received a message from the left of the wave at time T

Thus we must have a process q sending a message that crosses the wave (q = P3)

Visual demonstration of correctness

Assume all daemon counts match at the end



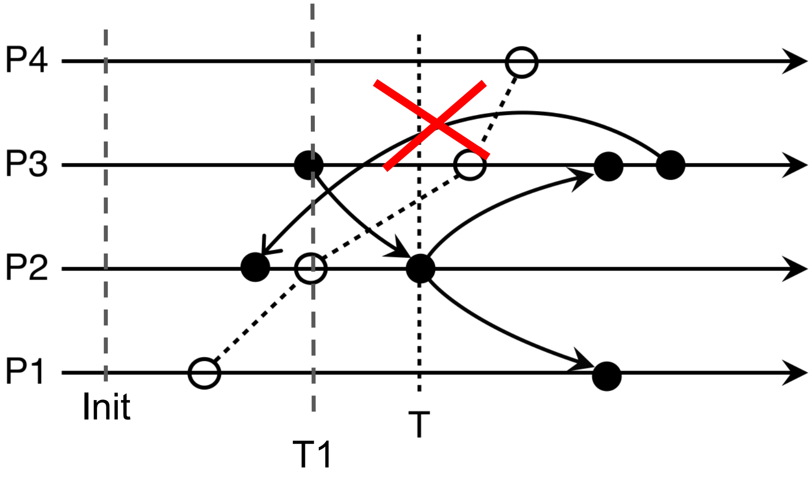
P received a message from the left of the wave

Thus we must have a process q sending a message that crosses the wave (q = P3)

Thus there must be a compensating messages crossing the wave in the other direction

Visual demonstration of correctness

Assume all daemon counts match at the end



P received a message from the left of the wave

Thus we must have a process q sending a message that crosses the wave (q = P3)

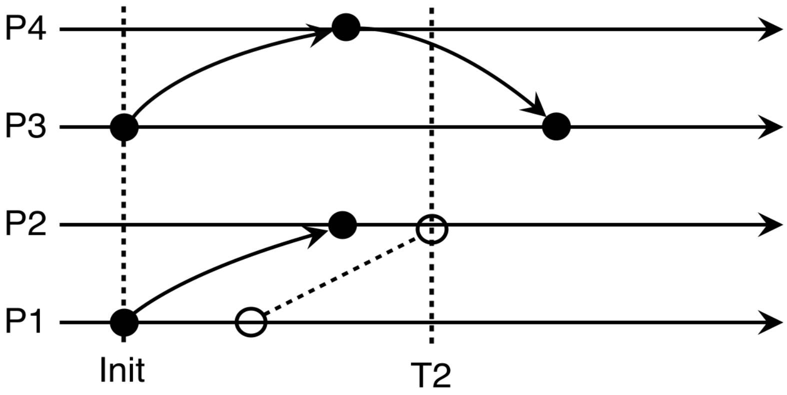
Thus there must be a compensating messages crossing the wave from right to left before T1

But messages cannot go back in time

Is this convincing?

Let’s at least check for safety violation in bounded executions with Apalache

Counterexample to safety



Counts match but the computation has not terminated

ds_{T2} =

	P1	P2	P3	P4
P1				
P2	1			
P3				
P4				

dr_{T2} =

	P1	P2	P3	P4
P1				
P2	1			
P3				
P4				

We must make sure we have visited all processes

Declare termination as soon as:

- the numbers collected so far are consistent, **and**
- all processes have been visited at least once

CONSTANT P

VARIABLES

$s,\ r,\ ds,\ dr,\ visited,\ terminated$

$Init \triangleq$

$\land\ s \in [P \times P \rightarrow Nat]$

$\land\ \exists\, p,\ q \in P : s[\langle p,\ q\rangle] > 0$ at least one message in flight

$\land\ r = [pq \in P \times P \mapsto 0]$

$\land\ ds = [pq \in P \times P \mapsto 0]$

$\land\ dr = [pq \in P \times P \mapsto 0]$

$\land\ visited = \{\}$

$\land\ terminated = \text{FALSE}$

$NumPending(p,\ q) \triangleq$

$s[\langle p,\ q\rangle] - r[\langle p,\ q\rangle]$

$Process(self) \triangleq$

$\land\ \exists\, p \in P \setminus \{self\} :$ receive a message from p

$\quad \land\ NumPending(p,\ self) > 0$

$\quad \land\ r' = [r \text{ EXCEPT } ![\langle p,\ self\rangle] = @ + 1]$

$\land\ \exists\, Q \in \text{SUBSET } (P \setminus \{self\}) :$ send messages to set Q

$\quad \land\ s' = [t \in P \times P \mapsto$

$\qquad \text{IF } t[1] = self \land t[2] \in Q \text{ THEN } s[t] + 1 \text{ ELSE } s[t]]$

$\land\ \text{UNCHANGED } \langle ds,\ dr,\ visited,\ terminated\rangle$

$Consistent(Q) \triangleq$

$\forall\, p,\ q \in Q : ds[\langle p,\ q\rangle] = dr[\langle q,\ p\rangle]$

$Daemon \triangleq$

$\land\ \text{IF } visited \neq P \lor \neg Consistent(P)$

$\quad \text{THEN } \exists\, p \in P :$ pick a process p to visit

$\qquad \land\ ds' = [t \in P \times P \mapsto \text{IF } t[1] = p \text{ THEN } s[t] \text{ ELSE } ds[t]]$

$\qquad \land\ dr' = [t \in P \times P \mapsto \text{IF } t[2] = p \text{ THEN } r[t] \text{ ELSE } dr[t]]$

$\qquad \land\ visited' = (visited \cup \{p\})$

$\qquad \land\ \text{UNCHANGED } terminated$

$\quad \text{ELSE}$

$\qquad \land\ terminated' = \text{TRUE}$ declare termination

$\qquad \land\ \text{UNCHANGED } \langle ds,\ dr,\ visited\rangle$

$\land\ \text{UNCHANGED } \langle s,\ r\rangle$

$Next \triangleq Daemon \lor (\exists\, p \in P : Process(p))$

$Safety \triangleq terminated \Rightarrow \forall\, p,\ q \in P : NumPending(p,\ q) = 0$

$Spec \triangleq Init \land \quad \Box[Next]_{vars}$

Is it convincing now?

Appeals to intuition, but:

- How do we systematically check we did not forget a case again?
- Can a program check that for us?

Formalize in Isabelle, Coq, Lean, etc.? Probably a lot of work

Instead, prove safety with an inductive invariants!

We just have to check validity of:

$$
\begin{aligned}
&\land\ Init \Rightarrow Inv \\
&\land\ Inv \land Next \Rightarrow Inv'
\end{aligned}
$$

Automated solvers can do it (at least for a fixed number of processes)

Main invariant (almost inductive)

If a set Q is consistent in the daemon's view but not in reality, it's because the daemon missed a message from outside Q to Q.

Why does this imply correctness?

Take Q=P.

It is not possible to receive a message from outside P.

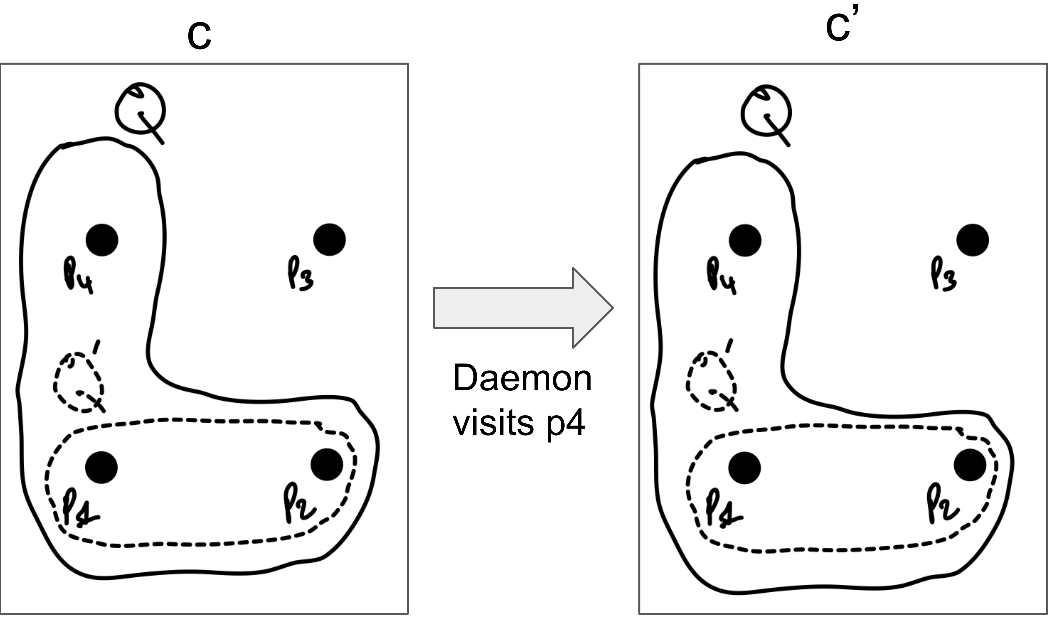
So when P is consistent in the daemon's view, it is also consistent in reality and the computation has terminated.

Full inductive invariant:

- If a set Q is consistent in the daemon's view but not in reality, it's because the daemon missed a message from outside Q to Q.
- Receive count on a channel is smaller than sent count

$$
\begin{array}{l}
Inv \triangleq \\
\quad \text{LET } Stale(Q) \triangleq \exists\, p \in Q,\; q \in P : \\
\qquad \lor\; r[\langle p, q \rangle] \neq dr[\langle p, q \rangle] \\
\qquad \lor\; s[\langle p, q \rangle] \neq ds[\langle p, q \rangle] \\
\quad \text{IN} \\
\qquad \land\; \forall\, p, q \in P : r[\langle p, q \rangle] \leq s[\langle p, q \rangle] \\
\qquad \land\; \forall\, Q \in \text{SUBSET } visited : Consistent(Q) \land Stale(Q) \\
\qquad\qquad \Rightarrow \exists\, p \in Q,\; q \in P \setminus Q : r[\langle p, q \rangle] > dr[\langle p, q \rangle]
\end{array}
$$

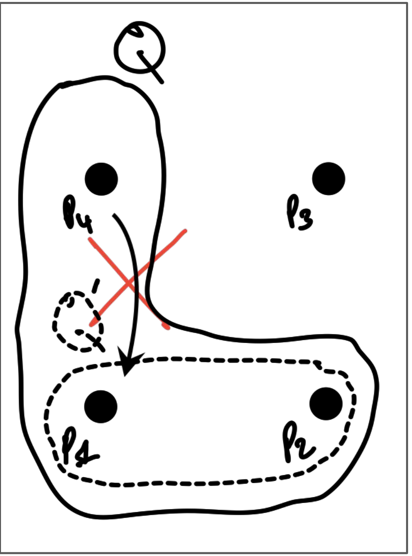
Take Q that is consistent but stale in c’
Show that the daemon missed a message from outside Q to Q



1. Q is consistent but stale in c’
2. Q’ is consistent but stale in c’
3. Q’ is consistent but stale in c

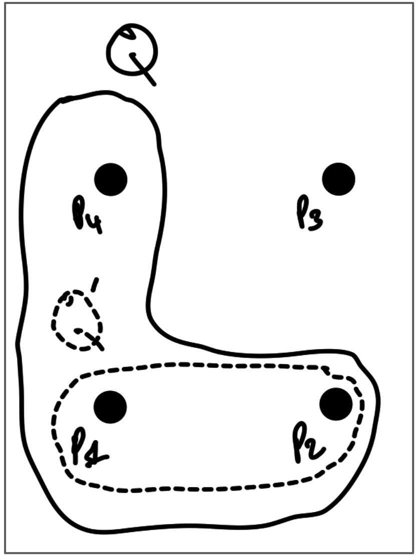
Take Q that is consistent but stale in c’
Show that the daemon missed a message from outside Q to Q

c



Daemon
visits p4

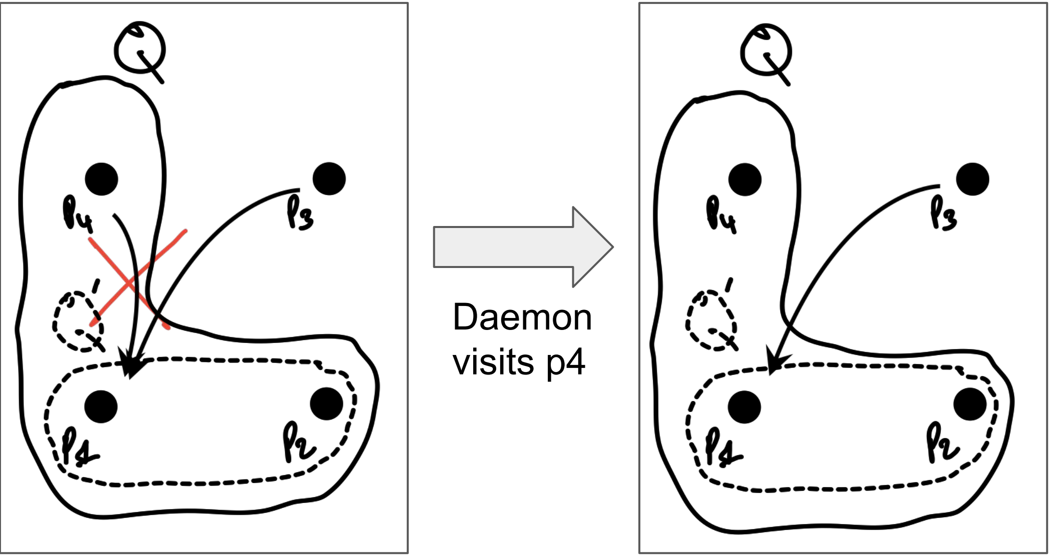
c’



1. Q is consistent but stale in c’
2. Q’ is consistent but stale in c’
3. Q’ is consistent but stale in c
4. By IH, he daemon missed a message to p1 from outside Q’
5. The message cannot be from p4

Take Q that is consistent but stale in c’
Show that the daemon missed a message from outside Q to Q

c c’



1. Q is consistent but stale in c’
2. Q’ is consistent but stale in c’
3. Q’ is consistent but stale in c
4. By IH, he daemon missed a message to p1 from outside Q’
5. The message cannot be from p4
6. Thus it’s from p3
7. QED

We check inductiveness by case analysis on a single step

$$\begin{aligned} &\land\ Init \Rightarrow Inv \\ &\land\ Inv \land Next \Rightarrow Inv' \end{aligned}$$

Not hard but a little tedious

Easy for automated tools, at least on bounded examples

- Apalache proves inductiveness for 6 processes and unbounded counts
- Mechanically-checked proof for any number of processes, in Isabelle/HOL: ~200 lines of proof

https://github.com/nano-o/Distributed-termination-detection

How do we find the invariant?

Interactive tools like Apalache help build intuition fast by providing lots of counter-examples.

Typical workflow:

1. Start with the correctness property
2. Get a counterexample to induction
3. Add a conjunct that eliminates the counterexample
4. Repeat

Step 3 requires some intuition to suitably generalize the counterexample

Inductive invariants are great!

- The “intuitive” proof is hard to check and easily leads to errors
- The inductive proof is easy to check and, with proper tooling, also feels intuitive
- Inductive invariants: A human-machine interface language
    - Plays the strength of both: the human uses intuition to come up with an inductive invariant; the machine enumerates all cases

Fun Exercises

1. The “wrong” algorithm is safe if we assume that a unique process sends the initial messages. Find an inductive invariant to prove this.
2. Proof with TLAPS?
3. Model and proof in Ivy
    a. Can we axiomatize the domain model (finite sets) in FOL and prove inductiveness for arbitrary system size automatically in Ivy?
    b. If not, where do we need higher-order reasoning? (Can also be done in Ivy by manually instantiating a higher-order axiom where needed)
4. Mechanize the “Wave” proof in a proof assistant? (Isabelle, Coq, Lean, etc.)

    Would it really be that hard?

Invariant: for every Q, if Q is consistent but stale in c’, then the daemon has missed a message from outside Q to Q.
Suppose c->c’ and the invariant holds in c. Fix Q that is consistent but stale in c’.
Show that, in c’, the daemon has missed a message from outside Q to Q.

- Suppose c->c’ is a receive step of process p.
  - Suppose Q is stale in c. Then, by induction hypothesis, in c the daemon missed a message as above. This remains true in c’.
  - Suppose Q is not stale in c. Then there are no messages in flight between members of Q. Moreover, p is in Q (other Q cannot change from not stale to stale). Thus p receives a message from outside Q.
- Suppose c->c’ is a step of the daemon.
  - Suppose the daemon sets the termination flag. Then counts don’t change.
  - Suppose the daemon updates the count of a process p.
    - Suppose p is not in Q. Then the counts affecting Q do not change.
    - Suppose p is in Q…

Invariant: for every Q, if Q is consistent but stale in c’, then the daemon has missed a message from outside Q to Q.
Suppose c->c’ and the invariant holds in c. Fix Q that is consistent but stale in c’.
Show that, in c’, the daemon has missed a message from outside Q to Q.

- Suppose c->c’ is a step of the daemon.
  - Suppose the daemon updates the count of a process p
    - Suppose p is in Q. Let Q’=Q\{p}.
      Because Q is consistent in c’, Q’ is consistent in c’ and so also in c.
      Because Q is stale in c’ and p’s counts cannot be stale, Q’ is stale in c’ and also in c.
      By induction hypothesis, in c, the daemon has has missed a message from outside Q’ to Q’.
      It now suffices to show that this message is not from p. This is the case because Q is consistent in c’, so any message from p to Q has been counted.