# Formalizing Vector Clocks in Agda (Functional Pearl)

ANONYMOUS AUTHOR(S)

Distributed systems are hard to build partly because the lack of physically synchronous global clocks makes reasoning about causality sometimes impossible. To this end, vector logical clocks have been proposed and proved to capture causality, in particular, they preserve and determin the happens-before relation. However, most of the proofs are done informally on paper.

In this paper...

## 1 INTRODUCTION

## 2 SYSTEM MODEL

We model a distributed system as consisting of a fixed number of processes communicating solely by messages. In Agda, we postulate the number of processes [1] and the type of the messages [2]:

postulate n : ℕ Message : Setpostulate n : ℕ Message : Set

In an abstract way, the history of a process can be viewed as a sequence of events that take place on it, where events are sendings and receivings of messages. Similarly, the history of one execution of a distributed system can be viewed as an collection of sequences of events of each process. We assume each process is assigned an unique identifier [3] and each message is assigned a local identifier that corresponds to the order it takes place on its originating process:

ProcessId = Fin n LocalEventId = ℕProcessId = Fin n LocalEventId = ℕ

As as result, we can identify a event by a product of its originating process's and its .

Events are related, one relation that we are in particular interested in is Lamport's happens-before relation that establishes an strict partial order on events:

**Definition 1** (Happens-Before)**.** We say event $e$ happens before $e'$ if and only if:

(1) $e$ and $e'$ occur on the same process and $e$ takes place before $e'$; or
(2) $e$ is a sending event and $e'$ is the correspoinding receiving event; or
(3) $e$ happens before $e''$ and $e''$ happens before $e'$ for any event $e''$.

Two distinct events $e$ and $e'$ are said to be concurrent if neither $e$ happens before or $e'$ happens before $e$. It is well-known that the happens-before relation captures the potential causality of events in an execution.

To decide if event $e$ happens before $e'$, one has to position the two events into a particular history and see if one of the three conditions of happens-before holds. This requires us to define reachable histories, which is traditionally done as a state transition system.

## 3 STRONG CLOCK CONDITION

### 3.1 Happens-Before Preserving

As a shorthand, we will refer to the index on the vector clock of an event $e$ that represents $e$'s process as $e$'s process index, i.e. $vc[e]$ 's value at $e$'s process index is $vc[e]pid[e]$. Then to prove *Happens-Before Determining*, $vc[\_]$ preserves the happens-before partial order between two events

(insert code for Happens-Before Determining)

We observe that definition of $vc[\_]$. makes sure that an event always increments its vector clock at its process index.

(insert definition of $vc[\_]$)

From this we can extract the following more facts, stated more verbosely:

- The vector clock of a send event *send m e* is greater than its local preceding event $e$ at their shared process index $pid[e]$, and equal to it at every other index.
- The vector clock of a receive event *recv e e'* is greater than the maximum $vc[e]$ of $vc[e']$ at its process index $pid[e']$, and equal to it at every other index. This in turns means it's greater than either $vc[e]$ of $vc[e']$ at $pid[e']$ and greater or equal to them at every other

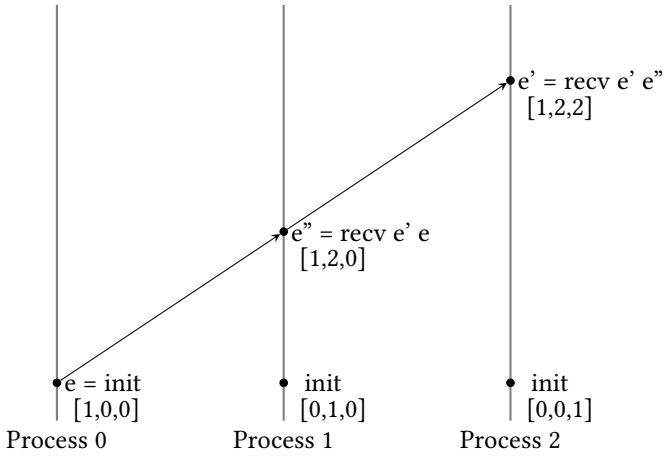


Fig. 1. Figure 1

not suffice since $e''$ and $e'$'s indices could be different. In this case, we invoke *Happens-Before Preserving* to establish that $vc[e]pid[e'] \leq vc[e'']pid[e']$, then induction on $e \sqsubset e'$ produces $vc[e]pid[e'] \leq vc[e'']pid[e'] < vc[e']pid[e']$. □

The second lemma is a special case of *Happens-Before Preserving* that operate on separate processes.

(insert code for Index Happens-Before Preserving)

Theorem 3.2 (Index Happens-Before Determining). *Let $e$ and $e'$ be two events on different processes, then $e$ is less than the vector clock of $e'$ at $e$'s index implies $e \sqsubset e'$*

(insert code for Index Happens-Before Determining)

The premises of this lemma is set up with structural induction on $e'$ in mind. Because $pid[e'] \not\equiv pid[e]$. It is often the case that $vc[e']pid[e]$ is unchanged from the preceding event, and we can resolve these cases using the inductive hypothesis. In fact, it is only when $e'$ is a *recv* event of another event $e''$ from $e$ 's process that this convenient induction scheme breaks. In this case, we could rely on the result in our tool box - the total order of events on $e$ and $e''$s shared process. If $e$ happens before or is equal to $e''$ (heterogenously due to our formation of the total order lemma), then we can derive $e \sqsubset e'$. The other case of $e'' \sqsubset e$ is impossible because by the first lemma, $vc[e'']pid[e''] < vc[e]pid[e'']$,which contradicts the premise $vc[e]pid[e] \leq vc[e'']pid[e'']$ □.

This lemma gives a $O(1)$ time check for whether an event $e$ happens before another event $e'$ from a different process - if $vc[e]pid[e] \leq vc[e']pid[e]$ then $e \sqsubset e'$ and if $vc[e']pid[e'] \leq vc[e]pid[e']$ then $e' \sqsubset e$, otherwise $e \| e'$.

Using the second lemma, we can prove *Happens-Before Determining* rather succinctly.

(insert code for Happens-Before Determining)

we again case split on whether $e$ and $e'$ are from the same process, i.e. $pid[e] \equiv pid[e']$. If they are not, then the first condition of the second lemma is satified , $vc[e] \prec vc[e']$ implies the second condition $vc[e]pid[e] < vc[e']pid[e]$, therefore we can directly apply our lemma to derive $e \sqsubset e'$.

Now we assume $e$ and $e'$ are from the same process. In this case, we use the fact that events on the same process form a total order, then either $e \sqsubset e'$, $e \approx e'$, or $e' \sqsubset e$. $e \cong e'$. $e \sqsubset e'$ is our end goal, and the other undesirable cases can be eliminated. If $e \cong e'$, then we can derive $vc[e] \cong vc[e']$ by congruence, meaning all of their indices are equal and contradicting $vc[e'] \prec vc[e]$. Similarly $e \sqsubset e'$

is also impossible because we can derive $vc[e'] \prec vc[e]$ by *Happens-Before Preserving*, contradicting the irreflexivity of $\prec$ □.

## 4 CONCLUSION

Vector clocks [**???**], blah blah blah.

## A PROOF OF INDEX HAPPENS-BEFORE PRESERVING

- If $e'$ is a *send* event following e, then $e$ and $e'$ are from the same process, and the vector clock is incremented at their shared index.
- If $e'$ is a *recv* event following $e$ or a *recv* event of $e$ , then the vector clock of $e$ is incorporated through pointwise maximum into the vector clock of $e'$, which then increments its own index
- If $e \sqsubset e'$ by transitivity, and then there is a hidden event $e''$, such that $e''$ happens after $e$ and before $e'$. Using only induction would not suffice since $e''$ and $e'$'s indices could be different. In this case, we invoke $\sqsubset -preserving$ to establish that $vc[e]pid[e'] \leq vc[e'']pid[e']$, then recursion on $e \sqsubset e'$ gives us $vc[e]pid[e'] \leq vc[e']pid[e']$.

## B PROOF OF INDEX HAPPENS-BEFORE DETERMINING

- If $e'$ is a *init* event - recall that the vector clock of an event starts from 0 in all but its own index, which is incremented to 1. Given $e$ and $e'$ are from different processes, we know $vc[e']pid[e] \equiv 0$, and the second premise then becomes $vc[e]pid[e] \leq 0$. However, the vector clock of e starts from 1 at its own in index never decreases. Therefore $vc[e]pid[e] \leq 0$ is impossible.
- If $e'$ is a *send* of another event $e''$ , then we know its vector clock value at $pid[e]$ is unchanged from $e''$, i.e. $vc[e]pid[e] \leq vc[e'']pid[e]$ and by induction $e \sqsubset e''$, then by transitivity $e \sqsubset e''$.
- If $e'$ is a *recv* of events $e''$ following $e'''$, then like before we know the vector clock value at $pid[e]$ is not incremented, and the premise becomes $vc[e]pid[e] \leq vc[e'']pid[e] \sqcap vc[e''']pid[e]$. Now consider which of $vc[e'']pid[e]$ and $vc[e''']pid[e]$ is greater.
  - If $vc[e''']pid[e]$ is greater, then $vc[e]pid[e] \leq vc[e''']pid[e]$, and by induction $e \sqsubset e'''$, and by transitivity $e \sqsubset e'$.
  - If $vc[e'']pid[e]$ is greater, then $vc[e]pid[e] \leq vc[e'']pid[e]$. We are not sure if $e''$ is from a different process than $e$, so both case need to be considered.
    - If it is different, then by induction $e \sqsubset e''$, and by transitivity $e \sqsubset e'$ as above.
    - If it is not different, we invoke the local total order of events on the same process. If $e \sqsubset e''$ or $e \cong e''$, then $e \sqsubset e'$ by *send* $\sqsubset$ *recv* and transitivity. The last case $e'' \sqsubset e$ is impossible as as applying *Index Happens-Before Preserving* gives us $vc[e'']pid[e] < vc[e]pid[e]$, contradicting the premise.