A case study on parametric verification of failure detectors

Thanh-Hai Tran[1], Igor Konnov[2], and Josef Widder[2]

[1] TU Wien, Austria
[2] Informal Systems, Austria

Abstract. Partial synchrony is a model of computation in many distributed algorithms and modern blockchains. Correctness of these algorithms requires the existence of bounds on message delays and on the relative speed of processes after reaching Global Stabilization Time (GST). This makes partially synchronous algorithms parametric in time bounds, which renders automated verification of partially synchronous algorithms challenging. In this paper, we present a case study on formal verification of both safety and liveness of a Chandra and Toueg failure detector that is based on partial synchrony. To this end, we specify the algorithm and the partial synchrony assumptions in three frameworks: TLA$^+$, Ivy, and counter automata. Importantly, we tune our modeling to use the strength of each method: (1) We are using counters to encode message buffers with counter automata, (2) we are using first-order relations to encode message buffers in Ivy, and (3) we are using both approaches in TLA$^+$. By running the tools for TLA$^+$ (TLC and APALACHE) and counter automata (FAST), we demonstrate safety for fixed time bounds. This helped us to find the inductive invariants for fixed parameters, which we used as a starting point for the proofs with Ivy. By running Ivy, we prove safety for arbitrary time bounds. Moreover, we show how to verify liveness of the failure detector by reducing the verification problem to safety verification. Thus, both properties are verified by developing inductive invariants with Ivy. We conjecture that correctness of other partially synchronous algorithms may be proven by following the presented methodology.

Keywords: Failure detectors · TLA$^+$ · Counter automata · FAST · Ivy.

1 Introduction

Distributed algorithms play a crucial role in modern infrastructure, but they are notoriously difficult to understand and to get right. Network topologies, message delays, faulty processes, the relative speed of processes, and fairness conditions might lead to behaviors that were neither intended nor anticipated by algorithm designers. Hence, many specification and verification techniques for distributed algorithms [22,27,29,13] have been developed.

Verification techniques for distributed algorithms usually focus on two models of computation: synchrony [32] and asynchrony [18,20]. Synchrony is hard to implement on real systems, while many basic problems in fault-tolerant distributed computing are unsolvable in asynchrony.

Partial synchrony lies between synchrony and asynchrony, and escapes their shortcomings. To guarantee liveness properties, proof-of-stake blockchains [8,35] and distributed algorithms [10,7] assume time constraints under partial synchrony. That is the existence of bounds Δ on message delay, and Φ on the relative speed of processes after some time point. This combination makes partially synchronous algorithms parametric in time bounds.While partial synchrony is important for system designers, it is challenging for verification.

We thus investigate verification of distributed algorithms under partial synchrony, and start with the specific class of failure detectors: a Chandra and Toueg failure detector [10]. This is a well-known algorithm under partial synchrony that provides a service that can be used to solve many problems in fault-tolerant distributed computing.

Contributions. In this paper, we do parametric verification of both safety and liveness of the Chandra and Toueg failure detector in case of unknown bounds Δ and Φ. In this case, both Δ and Φ are arbitrary, and the time constraints under partial synchrony hold from the beginning.

- To do that, we first extend the cutoff results in [34] for partial synchrony. In a nutshell, cutoff results allow us to verify a parameterized distributed algorithm by checking its fixed-size instances, instead of all instances [15]. By our cutoff results, we need to check only instances with two processes of the Chandra and Toueg failure detector under partial synchrony. The cutoff results in [34] are applicable to this failure detector under synchrony or asynchrony.
- Second, we introduce the encoding techniques to efficiently specify the failure detector based on our cutoff results. These techniques can tune our modeling to use the strength of the tools: FAST, Ivy, and model checkers for TLA$^+$.
- Then, we demonstrate how to reduce the liveness properties Eventually Strong Accuracy, and Strong Completeness to safety properties.
- Next, we check the safety property Strong Accuracy, and a safety one reduced from the liveness property Strong Completeness on instances with fixed parameters by using FAST, and model checkers for TLA$^+$.
- To verify cases of arbitrary bounds Δ and Φ, we find and prove inductive invariants of the failure detector with the interactive theorem prover Ivy. We reduce the liveness properties to safety properties by applying the mentioned techniques. While our specifications are not in the decidable theories that Ivy supports, Ivy requires no additional user assistance to prove most of our inductive invariants.

Related work. Research papers about partially synchronous algorithms, including papers about failure detectors [24,2,1] are mainly based on manual proofs, and do not usually provide formal specifications. Without these details, proving those distributed algorithms with interactive theorem provers [12,29] is impossible. Proving distributed algorithms typically requires inductive invariants; however, finding constraints in an inductive invariant is hard. When every parameter is fixed, system designers can try to do probabilistic random inductive invariant checking with TLA$^+$ and TLC [23]. System designers can use timed automata [3] and parametric verification frameworks [25,4,26] to specify and verify timed systems. In the context of timed systems, we are aware of only one paper about verification of failure detectors [5]. In this paper, the authors used three tools, namely UPPAAL [25], mCRL2 [9], and FDR2 [31] to verify small instances of a failure detector based on a logical ring arrangement of processes. Their verification approach required that message buffers were bounded, and had restricted behaviors in the specifications. Moreover, they did not consider the bound Φ on the relative speed of processes. In contrast, there are no restrictions on message buffers, and no ring topology in the Chandra and Toueg failure detector. Moreover, our work is to verify the Chandra and Toueg failure detector in case of arbitrary bounds. In recent years, automatic parameterized verification techniques [19,32,13] have been introduced for distributed systems, but they are designed for synchronous and/or asynchronous models. Hence, these techniques cannot cope with time constraints under partial synchrony. Interactive theorem provers have been used to prove correctness of distributed algorithms recently. For example, researchers proved safety of Tendermint consensus with Ivy [16].

Structure. In Section 2, we summarize the Chandra and Toueg failure detector, and the cutoff results in [34]. In Section 3, we extend the cutoff results in [34] for partial synchrony. Our encoding technique is presented in Section 4. Experiments for small Δ and Φ are described in Section 6. Ivy proofs for parametric Δ and Φ are discussed in Section 7.

2 Preliminaries

This section describes the Chandra and Toueg failure detector [10], and the cutoff results [34] that we can extend to this failure detector under partial synchrony.

A failure detector can be seen as an oracle to get information about crash failures in the distributed system. That failure detector usually guarantees some of the following properties [10]:

- Strong Accuracy: No process is suspected before it crashes.
  $$\mathbf{G}(\forall p, q \in 1..N\colon (\mathit{Correct}(p) \land \mathit{Correct}(q)) \Rightarrow \neg \mathit{Suspected}(p, q))$$
- Eventual Strong Accuracy: There is a time after which correct processes are not suspected by any correct process.
  $$\mathbf{F}\,\mathbf{G}(\forall p, q \in 1..N\colon (\mathit{Correct}(p) \land \mathit{Correct}(q)) \Rightarrow \neg \mathit{Suspected}(p, q))$$
- Strong Completeness: Eventually every crashed process is permanently suspected by every correct process.
  $$\mathbf{F}\,\mathbf{G}(\forall p, q \in 1..N\colon (\mathit{Correct}(p) \land \neg \mathit{Correct}(q)) \Rightarrow \mathit{Suspected}(p, q))$$

where $\mathbf{F}$ and $\mathbf{G}$ are operators in LTL (linear temporal logic), predicate $\mathit{Suspect}(p, q)$ refers to whether process p suspects process q in crashing, and predicate $\mathit{Correct}(p)$ refers to whether process p is correct. However, process p might crash later (and not recover). A crashed process p satisfies $\neg \mathit{Correct}(p)$.

Algorithm 1 presents the pseudo-code of the failure detector of [10]. A system instance has N processes that communicate with each other by sending-to-all

Algorithm 1 The eventually perfect failure detector algorithm in [10]

```
 1: Every process $p \in 1..N$ executes the following:
 2: for all $q \in 1..N$ do                                        ▷ Initalization step
 3:     $timeout[p, q] :=$ default-value
 4:     $suspected[p, q] := \bot$
 5: Send "alive" to all $q \in 1..N$                               ▷ Task 1: repeat periodically
 6: for all $q \in 1..N$ do                                        ▷ Task 2: repeat periodically
 7:     if $suspected[p, q] = \bot$ and not hear $q$ during last $timeout[p, q]$ ticks then
 8:         $suspected[p, q] := \top$
 9: if $suspected[p, q]$ then                     ▷ Task 3: when receive "alive" from $q$
10:     $timeout[p, q] := timeout[p, q] + 1$
11:     $suspected[p, q] := \bot$
```

and receiving messages through unbounded N^2 point-to-point communication channels. A process performs local computation based on received messages (we assume that a process also receives the messages that it sends to itself). In one system step, all processes may take up to one step. Some processes may crash, i.e., stop operating. Correct processes follow Algorithm 1 to detect crashes in the system. Initially, every correct process sets a default value for a timeout of each other, i.e. how long it should wait for others and assumes that no processes have crashed (Line 4). Every correct process p has three tasks: (i) repeatedly sends an "alive" message to all (Line 5), and (ii) repeatedly produces predictions about crashes of other processes based on timeouts (Line 6), and (iii) increases a timeout for process q if p has learned that its suspicion on q is wrong (Line 9). Notice that process p raises suspicion on the operation of process q (Line 6) by considering only information related to q: $timeout[p, q]$, $suspected[p, q]$, and messages that p has received from q recently.

In the asynchronous model, Algorithm 1 does not satisfy Eventually Strong Accuracy since there exists no bound on message delay, and messages sent by correct processes might always arrive after the timeout expires. Liveness of the failure detector is based on the existence of bounds Δ on the transmission delay of messages, and Φ on the relative speed of processes after reaching the global stabilization at some time point T_0 [10]. There are many models of partial synchrony [14,10], In this paper, we focus only on the case of unknown bounds Δ and Φ because other models might call for abstractions. In this case, $T_0 = 1$, and both parameters Δ and Φ are arbitrary. Moreover, the following time constraints hold in every execution:

– Constraint 1: If message m is placed in the message buffer from process q to process p by some $Send(m, p)$ at a time $s_1 \geq 1$, and if process p executes a $Receive(p)$ at a time s_2 with $s_2 \geq s_1 + \Delta$, then message m must be delivered to p at time s_2 or earlier.
– Constraint 2: In every contiguous time interval $[t, t + \Phi]$ with $t \geq 1$, every correct process must take at least one step.

These constraints make the failure detector parametric in Δ and Φ.

Moreover, Algorithm 1 is parameterized by the initial value of the timeout. If the timeout is set to a too-small initial value, messages sent by correct processes might always arrive after the initial timeout expires. It leads to a violation of the safety property Strong Accuracy.

In [34], Thanh-Hai et al. defined a class of symmetric point-to-point distributed algorithms that contains the failure detector [10], and proved cutoffs on the number of processes for this class under asynchrony. These cutoff results guarantee that analyzing instances with two processes is sufficient to reason about the correctness of all instances of the Chandra and Toueg failure detector under asynchrony. In the following section, we will generalize this result to partial synchrony, which allows us to verify the mentioned properties on the failure detector by checking instances with only two processes.

3 Cutoffs of the failure detector

In this section, we extend the cutoffs of symmetric point-to-point distributed algorithms in [34] for partial synchrony.

Notice that time parameters in partial synchrony only reduce the execution space compared to asynchrony. Hence, we can formalize the system behaviors under partial synchrony by extending the formalization of the system behaviors under asynchrony in [34] with the notion of time, message ages, time constraints under partial synchrony. (Our formalization is left for the report [33].)

Cutoffs In a nutshell, our cutoff results for the symmetric point-to-point class allow us to verify the mentioned properties on the failure detector under partial synchrony by checking small instances with one and/or two processes. Intuitively, the proofs of our cutoffs are based on the following observations:

- The global transition system and the desired property are symmetric [34].
- Let $\mathcal{G}_2$ and $\mathcal{G}_N$ be two instances of a symmetric point-to-point algorithm with 2 and N processes, respectively. By [34], two instances $\mathcal{G}_2$ and $\mathcal{G}_N$ are trace equivalent under a set of predicates in the desired property.
- We will now discuss that the constraints maintain partial synchrony. Let π_N be an execution in $\mathcal{G}_N$. We construct an execution π_2 in $\mathcal{G}_2$ by applying the index projection to π_N. The index projection is defined in [34]. Intuitively, the index projection discards processes $3..N$ as well as their corresponding messages and buffers. Moreover, for every $k, \ell \in \{1, 2\}$, the index projection preserves (i) at which point in time process k takes a step, and (ii) what action process k takes at a time $t \geq 0$, and (iii) messages from process k to process ℓ. For example, Figure 1 demonstrates an execution in $\mathcal{G}_2$ that is constructed based on a given execution in $\mathcal{G}_3$ with the index projection. Observe that Constraints 1 and 2 are maintained in this projection.
- Let π_2 be an execution in $\mathcal{G}_2$. We construct an execution π_N in $\mathcal{G}_N$ based on π_2 such that all processes $3..N$ crash from the beginning, and π_2 is an index projection of π_N [34]. For example, Figure 2 demonstrates an execution in $\mathcal{G}_3$ that is constructed based on an given execution in $\mathcal{G}_2$. If Constraints 1 and 2 hold on π_2, these constraints also hold on π_N.

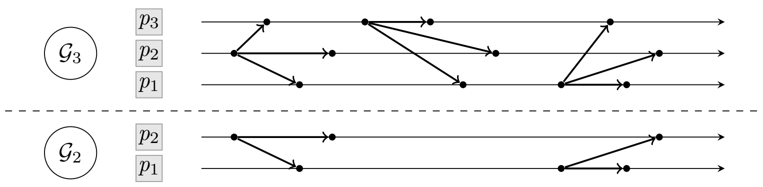

Fig. 1: Construct an execution in $\mathcal{G}_2$ based on a given execution in $\mathcal{G}_3$ by applying the index projection.

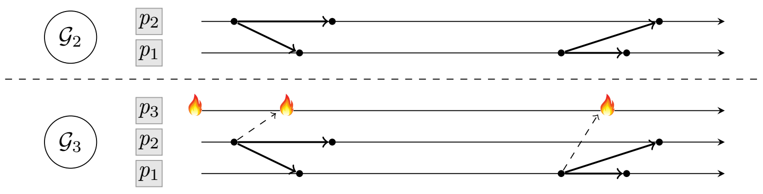

Fig. 2: Construct an execution in $\mathcal{G}_3$ based on a given execution in $\mathcal{G}_2$.

4 Encoding the Chandra and Toueg failure detector

In this section, we first discuss why it is sufficient to verify the failure detector by checking a system with only one sender and one receiver by applying the cutoffs presented in Section 3. Next, we introduce two approaches to encoding the message buffer, and an abstraction of in-transit messages that are older than Δ time-units. Finally, we present how to encode the relative speed of processes with counters over natural numbers. These techniques allow us to tune our models to the strength of the verification tools: FAST, Ivy, and model checkers for TLA$^+$.

4.1 The system with one sender and one receiver

We discussed the cutoff results in Section 3. These cutoff results allow us to verify the Chandra and Toueg failure detector under partial synchrony by checking only instances with two processes. From now on, we discuss the model with two processes, and formalize the properties with two process-indexes. In more detail, it is sufficient to verify Strong Accuracy, Eventually Strong Accuracy, and Strong Completeness by checking the following properties.

$$\mathbf{G}((\mathit{Correct}(1) \land \mathit{Correct}(2)) \Rightarrow \neg \mathit{Suspected}(2,1)) \tag{1}$$

$$\mathbf{F}\,\mathbf{G}((\mathit{Correct}(1) \land \mathit{Correct}(2)) \Rightarrow \neg \mathit{Suspected}(2,1)) \tag{2}$$

$$\mathbf{F}\,\mathbf{G}((\neg \mathit{Correct}(1) \land \mathit{Correct}(2)) \Rightarrow \mathit{Suspected}(2,1)) \tag{3}$$

We can take a further step towards facilitating verification of the failure detector. First, we can assume that each correct process never suspects itself. Sending messages to oneself is typically practiced with copying something in data structures, not really sending a message. Therefore, it makes sense to have this assumption. Second, local variables in Algorithm 1 are arrays whose elements correspond one-to-one with a remote process, e.g., *timeout*$[2, 1]$ and *suspected*$[2, 1]$. Third, communication between processes is point-to-point. When this is not the case, one can use cryptography to establish one-to-one communication. Hence, reasoning about Properties 1–3 requires no information about messages from process 1 to itself, and local variables of process 1, and messages from process 2.

Due to the above characteristics, it is sufficient to consider process 1 as a sender, and process 2 as a receiver. In detail, the sender follows Task 1 in Algorithm 1, but does nothing in Task 2 and Task 3. The sender does not need the initialization step, and local variables *suspected* and *timeout*. In contrast, the receiver has local variables corresponding to the sender, and follows only the initialization step, and Task 2, and Task 3 in Algorithm 1. The receiver can increase its waiting time in Task 1, but does not send any message.

4.2 Encoding the message buffer

Algorithm 1 assumes unbounded message buffers between processes that produce an infinite state space. Moreover, a sent message might be in-transit for a long time before it is delivered. In the following, we first introduce two approaches to encode the message buffer based on a logical predicate, and a counter over natural numbers. The first approach works for TLA$^+$ and Ivy, but not for counter automata (FAST). The latter is actually supported by all mentioned tools, but it is less efficient as it requires more transitions. Then, we present an abstraction of in-transit messages that are older than Δ time-units. This technique reduces the state space, and allows us to tune our models to the strength of the verification tools.

Encoding the message buffer with a predicate. In Algorithm 1, only "alive" messages are sent, and the message delivery depends only on the age of in-transit messages. Moreover, the computation of the receiver does not depend on the content of its received messages. Hence, we can encode a message buffer by using a logical predicate $\mathsf{existsMsgOfAge}(x)$. For every $k \geq 0$, predicate $\mathsf{existsMsgOfAge}(k)$ refers to whether there exists an in-transit message that is k time-units old. The number 0 refers to the age of a fresh message in the buffer.

It is convenient to encode the message buffer's behaviors in this approach. For instance, Formulas 4 and 5 show constraints on the message buffer when a new message is sent

$$\mathsf{existsMsgOfAge}'(0) = \top \tag{4}$$

$$\forall x \in \mathbb{N} \,.\, x > 0 \Rightarrow \mathsf{existsMsgOfAge}'(x) = \mathsf{existsMsgOfAge}(x) \tag{5}$$

where $\mathsf{existsMsgOfAge}'$ refers to the value of $\mathsf{existsMsgOfAge}$ in the next state. Formula 4 implies that a fresh message has been added to the message buffer. Formula 5 ensures that other in-transit messages are unchanged.

Another example is the relation between $\mathsf{existsMsgOfAge}$ and $\mathsf{existsMsgOfAge}'$ after the message delivery. This relation is formalized with Formulas 6–9. Formula 6 requires that there exists an in-transit message in $\mathsf{existsMsgOfAge}$ that can be delivered. Formula 7 ensures that no old messages are in transit after the delivery. Formula 8 guarantees that no message is created out of thin air. Formula 9 implies that at least one message is delivered.

$$\exists x \in \mathbb{N} \,.\, \mathsf{existsMsgOfAge}(x) \tag{6}$$
$$\forall x \in \mathbb{N} \,.\, x \geq \Delta \Rightarrow \neg \mathsf{existsMsgOfAge}'(x) \tag{7}$$
$$\forall x \in \mathbb{N} \,.\, \mathsf{existsMsgOfAge}'(x) \Rightarrow \mathsf{existsMsgOfAge}(x) \tag{8}$$
$$\exists x \in \mathbb{N} \,.\, \mathsf{existsMsgOfAge}'(x) \neq \mathsf{existsMsgOfAge}(x) \tag{9}$$

This encoding works for TLA$^+$ and Ivy, but not for FAST, because the input language of FAST does not support functions.

Encoding the message buffer with a counter. In the following, we present an encoding technique for the buffer that can be applied in all tools TLA$^+$, Ivy, and FAST. This approach encodes the message buffer with a counter `buf` over natural numbers. The k^{th} bit refers to whether there exists an in-transit message with k time-units old.

In this approach, message behaviors are formalized with operations in Presburger arithmetic. For example, assume $\Delta > 0$, we write $\mathtt{buf}' = \mathtt{buf} + 1$ to add a fresh message in the buffer. Notice that the increase of `buf` by 1 turns on the 0^{th} bit, and keeps the other bits unchanged.

To encode the increase of the age of every in-transit message by 1, we simply write $\mathtt{buf}' = \mathtt{buf} \times 2$. Assume that we use the least significant bit (LSB) first encoding, and the left-most bit is the 0^{th} bit. By multiplying `buf` by 2, we have updated $\mathtt{buf}'$ by shifting to the right every bit in `buf` by 1. For example, Figure 3 demonstrates the message buffer after the increase of message ages in case of $\mathtt{buf} = 6$. We have $\mathtt{buf}' = \mathtt{buf} \times 2 = 12$. It is easy to see that the 1^{st} and 2^{nd} bits in `buf` are on, and the 2^{nd} and 3^{rd} bits in $\mathtt{buf}'$ are on.

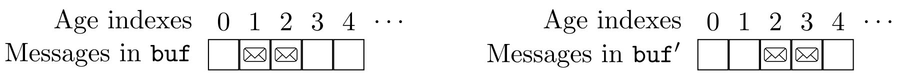

Fig. 3: The message buffer after increasing message ages in case of $\mathtt{buf} = 6$

Recall that Presburger arithmetic does not allow one to divide by a variable. Therefore, to guarantee the constraint in Formula 8, we need to enumerate all constraints on possible values of `buf` and $\mathtt{buf}'$ after the message delivery. For example, assume $\mathtt{buf} = 3$, and $\Delta = 1$. After the message delivery, $\mathtt{buf}'$ is either 0 or 1. If $\mathtt{buf} = 2$ and $\Delta = 1$, $\mathtt{buf}'$ must be 0 after the message delivery. Importantly, the number of transitions for the message delivery depends on the value of Δ.

To avoid the enumeration of all possible cases, Formula 8 can be rewritten with bit-vector arithmetic. However, bit-vector arithmetic are currently not supported in all verification tools TLA$^+$, FAST, and Ivy.

The advantage of this encoding is that when bound Δ is fixed, every constraint in the system behaviors can be rewritten in Presburger arithmetic. Thus, we can use FAST, which accepts constraints in Presburger arithmetic. To specify cases with arbitrary Δ, the user can use TLA$^+$ or Ivy.

Abstraction of old messages. Algorithm 1 assumes underlying unbounded message buffers between processes. Moreover, a sent message might be in transit for a long time before it is delivered. To reduce the state space, we develop an abstraction of in-transit messages that are older than Δ time-units; we call such messages "old". This abstraction makes the message buffer between the sender and the receiver bounded. In detail, the message buffer has a size of Δ. Importantly, we can apply this abstraction to two above encoding techniques for the message buffer.

In partial synchrony, if process p executes Receive at some time point from the Global Stabilization Time, *every* old message sent to p will be delivered immediately. Moreover, the computation of a process in Algorithm 1 does not depend on the content of received messages. Hence, instead of tracking all old messages, our abstraction keeps only one old message that is Δ time-units old, does not increase its age, and throws away other old messages.

In the following, we discuss how to integrate this abstraction into the encoding techniques of the message buffer. We demonstrate our ideas by showing the pseudo-code of the increase of message ages. It is straightforward to adopt this abstraction to the message delivery, and to the sending of a new message.

Figure 4(a) presents the increase of message ages with this abstraction in a case of $\Delta = 2$, and `buf` $= 6$. Unlike Figure 3, there exists no in-transit message that is 3 time-units old in Figure 4(a). Moreover, the message buffer in Figure 4(a) has a size of 3. In addition, `buf'` has only one in-transit message that is 2 time-units old. We have `buf'` $= 4$ in this case. Figure 4(b) demonstrates another case of $\Delta = 2$, `buf` $= 5$, and `buf'` $= 6$.

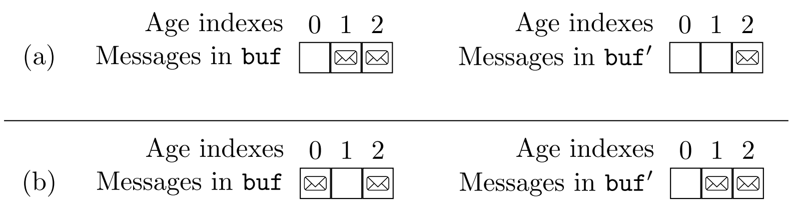

Fig. 4: The increase of message ages with the abstraction of old messages. In the case (a), we have $\Delta = 2$, `buf` $= 6$, and `buf'` $= 4$. In the case (b), we have $\Delta = 2$, `buf` $= 5$, and `buf'` $= 6$.

Formally, Figure 5 presents the pseudo-code of the increase of message ages that is encoded with a counter `buf`, and the abstraction of old messages. There are three cases. In the first case (Line 1), there exist no old messages in `buf`, and we simply set $\texttt{buf}' = \texttt{buf} \times 2$. In other cases (Lines 3 and 4), `buf` contains an old message. Figure 4(a) demonstrates the second case (Line 3). We subtract $2^{\Delta+1}$ to remove an old message with $\Delta + 1$ time-units old from the buffer. Figure 4(b) demonstrates the third case (Line 4). In the third case, we also need to remove an old message with $\Delta + 1$ time-units old from the buffer. Moreover, we need to put an old message with Δ time-units old to the buffer by adding 2^{Δ}.

```
1: if $\texttt{buf} < 2^{\Delta}$ then $\texttt{buf}' \leftarrow \texttt{buf} \times 2$
2: else
3:     if $\texttt{buf} \geq 2^{\Delta} + 2^{\Delta-1}$ then $\texttt{buf}' \leftarrow \texttt{buf} \times 2 - 2^{\Delta+1}$
4:     else $\texttt{buf}' \leftarrow \texttt{buf} \times 2 - 2^{\Delta+1} + 2^{\Delta}$
```

Fig. 5: Encoding the increase of message ages with a counter `buf`, and the abstraction of old messages.

Now we discuss how to integrate the abstraction of old messages in the encoding of the message buffer with a predicate. Formulas 10–13 present the relation between existsMsgOfAge and existsMsgOfAge′ when message ages are increased by 1, and this abstraction is applied. Formula 10 ensures that no fresh message will be added to existsMsgOfAge′. Formula 11 ensures that the age of every message that is until $(\Delta - 2)$ time-units old will be increased by 1. Formulas 12–13 are introduced by this abstraction. Formula 12 implies that if there exists an old message or a message with $(\Delta - 1)$ time-units old in existsMsgOfAge, there will be an old message that is Δ time-units old in existsMsgOfAge′. Formula 13 ensures that there exists no message that is older than Δ time-units old.

$$\neg \mathsf{existsMsgOfAge}'(0) \tag{10}$$

$$\forall x \in \mathbb{N} \,.\, (0 \leq x \leq \Delta - 2) \Rightarrow \mathsf{existsMsgOfAge}'(x+1) = \mathsf{existsMsgOfAge}(x) \tag{11}$$

$$\mathsf{existsMsgOfAge}'(\Delta) = \mathsf{existsMsgOfAge}(\Delta) \vee \mathsf{existsMsgOfAge}(\Delta - 1) \tag{12}$$

$$\forall x \in \mathbb{N} \,.\, x > \Delta \Rightarrow \mathsf{existsMsgOfAge}'(x) = \bot \tag{13}$$

4.3 Encoding the relative speed of processes

Recall that we focus on the case of unknown bounds Δ and Φ. In this case, every correct process must take at least one step in every contiguous time interval containing Φ time-units [14].

To maintain this constraint on executions generated by the verification tools, we introduced two additional control variables sTimer and rTimer for the sender and the receiver, respectively. These variables work as timers to keep track of how long a process has not taken a step, and when a process can take a step. Since these timers play similar roles, we here focus on rTimer. In our encoding, only the environment can update rTimer. To schedule the receiver, the environment non-deterministically executes one of two actions: (i) resets rTimer to 0, and (ii) if $\mathsf{rTimer} < \Phi$, increases rTimer by 1. Moreover, the receiver must take a step whenever $\mathsf{rTimer} = 0$.

5 Reduce liveness properties to safety properties

To verify the liveness properties Eventually Strong Accuracy and Strong Completeness with Ivy, we first need to reduce them to safety properties. Intuitively, these liveness properties are bounded; therefore, they become safety ones. This section demonstrates how to reduce Eventually Strong Accuracy to a safety one.

By cutoffs discussed in Section 3, it is sufficient to verify Eventually Strong Accuracy on the Chandra and Toueg failure detector by checking the following property on instances with 2 processes.

$$\mathbf{F}\,\mathbf{G}((\mathit{Correct}(1) \land \mathit{Correct}(2)) \Rightarrow \neg \mathit{Suspected}(2,1)) \tag{14}$$

In the failure detector [10], the receiver suspects the sender only if its waiting time reaches the timeout (see Line 6 in Algorithm 1). To reduce Formula 14 to a safety property, we found a specific guard g for $\mathsf{timeout}$ such that if $\mathsf{timeout} \geq g$ and the sender is correct, then $\mathtt{waitingtime} < g$. Hence, it is sufficient to verify Formula 14 by checking the following property.

$$\mathbf{G}\left(\mathsf{timeout} \geq g \Rightarrow ((\mathit{Correct}(1) \land \mathit{Correct}(2)) \Rightarrow \neg \mathit{Suspected}(2,1))\right)$$

6 Experiments for small Δ and Φ

In this section, we describe our experiments with TLA$^+$ and FAST. We ran the following experiments on a virtual machine with Core i7-6600U CPU and 8GB DDR4. Our specifications can be found at [33].

6.1 Model checkers for TLA$^+$: TLC and APALACHE

In our work, we use TLA$^+$ [22] to specify the failure detector with both encoding techniques for the message buffer, and the abstraction in Section 4. Then, we use the model checkers TLC [36] and APALACHE [17] to verify instances with fixed bounds Δ and Φ, and the GST $T_0 = 1$. This approach helps us to search constraints in inductive invariants in case of fixed parameters. The main reason is that counterexamples and inductive invariants in case of fixed parameters, e.g., $\Delta \leq 1$ and $\Phi \leq 1$, are simpler than in case of arbitrary parameters. Hence, if a counterexample is found, we can quickly analyze it, and change constraints in an inductive invariant candidate. After obtaining inductive invariants in small cases, we can generalize them for cases of arbitrary bounds, and check with theorem provers, e.g., Ivy (Section 7).

TLA$^+$ offers a rich syntax for sets, functions, tuples, records, sequences, and control structures [22]. Hence, it is straightforward to apply the encoding techniques and the abstraction presented in Section 4 in TLA$^+$. For example, Figure 6 represents a TLA$^+$ action *SSnd* for sending a new message in case of $\Delta > 0$. Variables *ePC* and *sPC* are program counters for the environment and the sender, respectively. Line 1 is a precondition, and refers to that the environment is in subround Send. Lines 2–3 say that if the sender is active in subround Send, the counter buf' is increased by 1. Otherwise, two counters buf and buf' are the same (Line 4). Line 5 implies that the environment is still in the subround Send, but it is now the receiver's turn. Line 6 guarantees that other variables are unchanged in this action. (The details are left for the report [33].)

```
1: $SSnd \triangleq$ $\land\ ePC = $ "SSnd"
2:                  $\land$ IF $(sTimer = 0 \land sPC = $ "SSnd"$)$
3:                       THEN $buf' = buf + 1$
4:                       ELSE UNCHANGED $buf$
5:                  $\land\ ePC' = $ "RNoSnd"
6:                  $\land$ UNCHANGED $\langle sTimer, rTimer ... \rangle$
```

Fig. 6: Sending a new message in TLA$^+$ in case of $\Delta > 0$

Now we present the experiments with TLC and APALACHE. We used these tools to verify (i) the safety property Strong Accuracy, and (ii) an inductive invariant for Strong Accuracy, and (iii) an inductive invariant for a safety property reduced from the liveness property Strong Completeness in case of fixed bounds, and GST = 1 (initial stabilization). The inductive invariants verified here are very close to the inductive invariants in case of arbitrary bounds Δ and Φ.

Table 1 shows the results in verification of Strong Accuracy in case of the initial stabilization, and fixed bounds Δ and Φ. Table 1 shows the experiments with the three tools TLC, APALACHE, and FAST. The column "#states" shows the number of distinct states explored by TLC. The column "#depth" shows the maximum execution length reached by TLC and APALACHE. The column "buf" shows how to encode the message buffer. The column "LOC" shows the number of lines in the specification of the system behaviors (without comments). The symbol "-" (minus) refers to that the experiments are intentionally missing since FAST does not support the encoding of the message buffer with a predicate. The abbreviation "pred" refers to the encoding of the message buffer with a predicate. The abbreviation "cntr" refers to the encoding of the message buffer with a counter. The abbreviation "TO" means a timeout of 6 hours. In these experiments, we initially set $\mathsf{timeout} = 6 \times \Phi + \Delta$, and Strong Accuracy is satisfied. The experiments show that TLC finishes its tasks faster than the others, and APALACHE prefers the encoding of the message buffer with a predicate.

Table 2 summarizes the results in verification of Strong Accuracy with the tools TLC, APALACHE, and FAST in case of the initial stabilization, and small bounds Δ and Φ, and initially $\mathsf{timeout} = \Delta + 1$. Since $\mathsf{timeout}$ is initialized with a too small value, there exists a case in which sent messages are delivered after the timeout expires. The tools reported an error execution where Strong Accuracy is violated. In these experiments, APALACHE is the winner. The abbreviation "TO" means a timeout of 6 hours. The meaning of other columns and abbreviations is the same as in Table 1.

Table 1: The experiments on checking Strong Accuracy with TLC, APALACHE, and FAST in case of fixed parameters. In this case, Strong Accuracy is satisfied.

#	Δ	Φ	buf	TLC				APALACHE		FAST	
				time	#states	depth	LOC	time	depth	time	LOC
1	2	4	pred	3s	10.2K	176	190	8m	176	-	-
2			cntr	3s	10.2K	176	266	9m	176	16m	387
3	4	4	pred	3s	16.6K	183	190	12m	183	-	-
4			cntr	3s	16.6K	183	487	35m	183	TO	2103
5	4	5	pred	3s	44.7K	267	190	TO	222	-	-
6			cntr	3s	44.7K	267	487	TO	223	TO	2103

Table 2: The experiments on checking Strong Accuracy with TLC, APALACHE, and FAST in case of fixed parameters. In this case, Strong Accuracy is violated.

#	Δ	Φ	buf	TLC			APALACHE		FAST
				time	#states	depth	time	depth	time
1	2	4	pred	1s	840	43	11s	42	-
2			cntr	1s	945	43	12s	42	10m
3	4	4	pred	2s	1.3K	48	15s	42	-
4			cntr	2s	2.4K	56	16s	42	TO
5	20	20	pred	TO	22.1K	77	1h15m	168	-

Table 3 shows the results in verification of inductive invariants for Strong Accuracy and Strong Completeness with TLC and APALACHE in case of the initial stabilization, and slightly larger bounds Δ and Φ. The message buffer was encoded with a predicate in these experiments. In these experiments, inductive invariants hold, and APALACHE is faster than TLC in verifying them.

As one sees from the tables, APALACHE is fast at proving inductive invariants, and at finding a counterexample when a desired safety property is violated. TLC is a better option in cases where a safety property is satisfied.

In order to prove correctness of the failure detector in cases where parameters Δ and Φ are arbitrary, the user can use the interactive theorem prover TLA$^+$ Proof System (TLAPS) [11]. A shortcoming of TLAPS is that it does not provide a counterexample when an inductive invariant candidate is violated. Moreover, proving the failure detector with TLAPS requires more human effort than with Ivy. Therefore, we provide Ivy proofs in Section 7.

Table 3: The experiments on checking inductive invariants with the tools TLC and APALACHE. In this case, inductive invariants are satisfied.

#	Δ	Φ	Property	TLC		APALACHE
				time	#states	time
1	4	40	Strong Accuracy	33m	347.3M	12s
2	4	10	Strong Completeness	44m	13.4M	17s

6.2 FAST

A shortcoming of the model checkers TLC and APALACHE is that parameters Δ and Φ must be fixed before running these tools. FAST is a tool designed to reason about safety properties of counter systems, i.e. automata extended with unbounded integer variables [6]. If Δ is fixed, and the message buffer is encoded with a counter, the failure detector becomes a counter system. We specified the failure detector in FAST, and made experiments with different parameter values to understand the limit of FAST: (i) the initial stabilization, and small bounds Δ and Φ, and (ii) the initial stabilization, fixed Δ, but unknown Φ.

Figure 7 represents a FAST transition for sending a new message in case of $\Delta > 0$. Line 2 describes the (symbolic) source state of the transition, and region `incMsgAge` is a set of configurations in the failure detector that is reachable from a transition for increasing message ages. Line 3 mentions the (symbolic) destination state of the transition, and region `sSnd` is a set of configurations in the failure detector that is reachable from a transition named "SSnd_Active" for sending a new message. Line 4 represents the guard of this transition. Line 5 is an action. Every unprimed variable that is not written in Line 5 is unchanged.

```
1: transition SSnd_Active := {
2:       from := incMsgAge;
3:       to := ssnd;
4:       guard := sTimer = 0;
5:       action := buf′ = buf + 1; };
```

Fig. 7: Sending a new message in FAST in case of $\Delta > 0$

The input language of FAST is based on Presburger arithmetics for both system and properties specification. Hence, we cannot apply the encoding of the message buffer with a predicate in FAST.

Tables 1 and 2 described in the previous subsection summarize the experiments with FAST, and other tools where all parameters are fixed. Moreover, we ran FAST to verify Strong Accuracy in case of the initial stabilization, $\Delta \leq 4$, and arbitrary Φ. FAST is a semi-decision procedure; therefore, it does not terminate on some inputs. Unfortunately, FAST could not prove Strong Accuracy in case of arbitrary Φ, and crashed after 30 minutes.

7 Ivy proofs for parametric Δ and Φ

While TLC, APALACHE, and FAST can automatically verify some instances of the failure detector with fixed parameters, these tools cannot handle cases with unknown bounds Δ and Φ. To overcome this problem, we specify and prove correctness of the failure detector with the interactive theorem prover Ivy [28]. In the following, we first discuss the encoding of the failure detector, and then presents the experiments with Ivy.

The encoding of the message buffer with a counter requires that bound Δ is fixed. We here focus on cases where bound Δ is unknown. Hence, we encode the message buffer with a predicate in our Ivy specifications,

In Ivy, we declare `relation` $\mathsf{existsMsgOfAge}(X : \mathsf{num})$. Type `num` is interpreted as integers. Since Ivy does not support primed variables, we need an additional relation $\mathsf{tmpExistsMsgOfAge}(X : \mathsf{num})$. Intuitively, we first compute and store the value of $\mathsf{existsMsgOfAge}$ in the next state in $\mathsf{tmpExistsMsgOfAge}$, then copy the value of $\mathsf{tmpExistsMsgOfAge}$ back to $\mathsf{existsMsgOfAge}$. We do not consider the requirement of $\mathsf{tmpExistsMsgOfAge}$ as a shortcoming of Ivy since it is still straightforward to transform the ideas in Section 4 to Ivy.

Figure 8 represents how to add a fresh message in the message buffer in Ivy. Line 1 means that $\mathsf{tmpExistsMsgOfAge}$ is assigned an arbitrary value. Line 2 guarantees the appearance of a fresh message. Line 3 ensures that every in-transit message in $\mathsf{existsMsgOfAge}$ is preserved in $\mathsf{tmpExistsMsgOfAge}$. Line 4 copies the value of $\mathsf{tmpExistsMsgOfAge}$ back to $\mathsf{existsMsgOfAge}$.

```
1: $\mathsf{tmpExistsMsgOfAge}(X) := *$;
2: **assume** $\mathsf{tmpExistsMsgOfAge}(0)$;
3: **assume forall** $X : \mathsf{num} \,.\, 0 < X \rightarrow \mathsf{existsMsgOfAge}(X) = \mathsf{tmpExistsMsgOfAge}(X)$;
4: $\mathsf{existsMsgOfAge}(X) := \mathsf{tmpExistsMsgOfAge}(X)$;
```

Fig. 8: Adding a fresh message in Ivy

Importantly, our specifications are not in decidable theories supported by Ivy. In Formula 11, the interpreted function " $+$ " (addition) is applied to a universally quantified variable x.

The standard way to check whether a safety property *Prop* holds in an Ivy specification is to find an inductive invariant *IndInv* with *Prop*, and to (interactively) prove that *Indinv* holds in the specification. To verify the liveness properties Eventually Strong Accuracy, and Strong Completeness, we reduced them into safety properties by applying a reduction technique in Section 5, and found inductive invariants containing the resulted safety properties. These inductive invariants are the generalization of the inductive invariants in case of fixed parameters that were found in the previous experiments.

Table 4 shows the experiments on verification of the failure detector with Ivy in case of unknown Δ and Φ. The symbol $\star$ refers to that the initial value of $\mathsf{timeout}$ is arbitrary. The column "#line$_I$" shows the number of lines of an inductive invariant, and the column "#strengthening steps" shows the number of lines of strengthening steps that we provided for Ivy. The meaning of other columns is the same as in Table 1. While our specifications are not in the decidable theories supported in Ivy, our experiments show that Ivy needs no user-given strengthening steps to prove most of our inductive invariants. Hence, it took us about 4 weeks to learn Ivy from scratch, and to prove these inductive invariants.

The most important thing to prove a property satisfied in an Ivy specification is to find an inductive invariant. Our inductive invariants use non-linear integers, quantifiers, and uninterpreted functions. (The inductive invariants in Table 4 are given in the report [33].)

Table 4: The experiments on verification of inductive invariants of the failure detector with Ivy in case of arbitrary $\varDelta$ and $\varPhi$.

#	Property	$\mathsf{timeout}_{\text{init}}$	time	LOC	#line$_I$	#strengthening steps
1	Strong Accuracy	$= 6 \times \varPhi + \varDelta$	4s	183	30	0
2	Eventually Strong Accuracy	$= \star$	4s	186	35	0
3	Strong Completeness	$= 6 \times \varPhi + \varDelta$	8s	203	111	0
4		$\geq 6 \times \varPhi + \varDelta$	22s	207	124	15
5		$= \star$	44s	207	129	0

8 Conclusion

We have presented verification of both safety and liveness of the Chandra and Toeug failure detector by using the verification tools: model checkers for TLA$^+$ (TLC and APALACHE), counter automata (FAST), and the theorem prover Ivy. To do that, we first prove the cutoff results that can apply to the failure detector under partial synchrony. Next, we develop the encoding techniques to efficiently specify the failure detector, and to tune our models to the strength of the mentioned tools. We verified safety in case of fixed parameters by running the tools TLC, APALACHE, and FAST. To cope with cases of arbitrary bounds $\varDelta$ and $\varPhi$, we reduced liveness properties to safety properties, and proved inductive invariants with desired properties in Ivy. While our specifications are not in the decidable theories supported in Ivy, our experiments show that Ivy needs no additional user assistance to prove most of our inductive invariants.

We found that modeling the failure detector in TLA$^+$ is helpful in understanding and finding inductive invariants in case of fixed parameters. These inductive invariants are simpler but very closed to ones in case of arbitrary parameters.

We found that features in the TLA$^+$ Toolbox [21], e.g., Profiler and Trace Exploration, are useful to analyze execution traces and returned counterexamples. One of Ivy's strong points is that it usually produces a counterexample quickly when a property is violated, even if all parameters are arbitrary. In contrast, FAST reports no counterexample in any case. Hence, debugging in FAST is very challenging.

While our specification describes executions of the Chandra and Toueg failure detector, we conjecture that many time constraints on network behaviors, correct processes, and failures in our inductive invariants can be reused to prove other algorithms under partial synchrony. We also conjecture that correctness of other partially synchronous algorithms may be proven by following the presented methodology. For future work, we would like to extend the above results for cases where GST is arbitrary. It is also interesting to investigate how to express discrete partial synchrony in timed automata [3], e.g., UPPAAL [25].

References

1. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: On implementing omega in systems with weak reliability and synchrony assumptions. Distributed Computing **21**(4), 285–314 (2008)
2. Aguilera, M.K., Delporte-Gallet, C., Fauconnier, H., Toueg, S.: Consensus with Byzantine failures and little system synchrony. In: DSN. pp. 147–155. IEEE (2006)
3. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical computer science **126**(2), 183–235 (1994)
4. André, É., Fribourg, L., Kühne, U., Soulat, R.: IMITATOR 2.5: A tool for analyzing robustness in scheduling problems. In: FM. pp. 33–36. Springer (2012)
5. Atif, M., Mousavi, M.R., Osaiweran, A.: Formal verification of unreliable failure detectors in partially synchronous systems. In: SAC. pp. 478–485 (2012)
6. Bardin, S., Leroux, J., Point, G.: Fast extended release. In: CAV. pp. 63–66 (2006)
7. Bravo, M., Chockler, G., Gotsman, A.: Making Byzantine consensus live. In: DISC. Schloss Dagstuhl-Leibniz-Zentrum für Informatik (2020)
8. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. arXiv preprint arXiv:1807.04938 (2018)
9. Bunte, O., Groote, J.F., Keiren, J.J., Laveaux, M., Neele, T., de Vink, E.P., Wesselink, W., Wijs, A., Willemse, T.A.: The mCRL2 toolset for analysing concurrent systems. In: TACAS. pp. 21–39. Springer (2019)
10. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. Journal of the ACM **43**(2), 225–267 (1996)
11. Chaudhuri, K., Doligez, D., Lamport, L., Merz, S.: The TLA$^+$ Proof System: Building a heterogeneous verification platform. In: ICTAC. pp. 44–44. Springer (2010)
12. Cousineau, D., Doligez, D., Lamport, L., Merz, S., Ricketts, D., Vanzetto, H.: TLA$^+$ proofs. In: FM. pp. 147–154. Springer (2012)
13. Drăgoi, C., Widder, J., Zufferey, D.: Programming at the edge of synchrony. Proceedings of the ACM on Programming Languages **4**(OOPSLA), 1–30 (2020)
14. Dwork, C., Lynch, N., Stockmeyer, L.: Consensus in the presence of partial synchrony. Journal of the ACM **35**(2), 288–323 (1988)
15. Emerson, E.A., Namjoshi, K.S.: Reasoning about rings. In: POPL. pp. 85–94 (1995)
16. Galois, I.: Ivy proofs of tendermint, URL: `https://github.com/tendermint/spec/tree/master/ivy-proofs`, accessed: December 2020
17. Konnov, I., Kukovec, J., Tran, T.H.: TLA$^+$ model checking made symbolic. Proceedings of the ACM on Programming Languages **3**(OOPSLA), 1–30 (2019)
18. Konnov, I., Lazic, M., Veith, H., Widder, J.: Para^2: Parameterized path reduction, acceleration, and SMT for reachability in threshold-guarded distributed algorithms. Formal Methods in System Design **51**(2), 270–307 (2017)
19. Konnov, I., Lazić, M., Veith, H., Widder, J.: Para^2: parameterized path reduction, acceleration, and SMT for reachability in threshold-guarded distributed algorithms. Formal Methods in System Design **51**(2), 270–307 (2017)
20. Konnov, I., Lazić, M., Veith, H., Widder, J.: A short counterexample property for safety and liveness verification of fault-tolerant distributed algorithms. In: POPL. pp. 719–734 (2017)
21. Kuppe, M.A., Lamport, L., Ricketts, D.: The TLA$^+$ toolbox. arXiv preprint arXiv:1912.10633 (2019)
22. Lamport, L.: Specifying systems: The TLA$^+$ language and tools for hardware and software engineers. Addison-Wesley (2002)

23. Lamport, L.: Using tlc to check inductive invariance (2018)
24. Larrea, M., Arévalo, S., Fernndez, A.: Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In: DISC. pp. 34–49. Springer (1999)
25. Larsen, K.G., Pettersson, P., Yi, W.: UPPAAL in a nutshell. International Journal on Software Tools for Technology Transfer **1**(1-2), 134–152 (1997)
26. Lime, D., Roux, O.H., Seidner, C., Traonouez, L.M.: Romeo: A parametric model-checker for Petri nets with stopwatches. In: TACAS. pp. 54–57. Springer (2009)
27. Lynch, N.A., Tuttle, M.R.: An introduction to input/output automata. Laboratory for Computer Science, Massachusetts Institute of Technology (1988)
28. McMillan, K.L.: Ivy, URL: https://microsoft.github.io/ivy/, accessed: December 2020
29. McMillan, K.L., Padon, O.: Ivy: a multi-modal verification tool for distributed algorithms. In: CAV. pp. 190–202. Springer (2020)
30. Padon, O., McMillan, K.L., Panda, A., Sagiv, M., Shoham, S.: Ivy: safety verification by interactive generalization. In: PLDI. pp. 614–630 (2016)
31. Roscoe, A.W.: Understanding concurrent systems. Springer Science & Business Media (2010)
32. Stoilkovska, I., Konnov, I., Widder, J., Zuleger, F.: Verifying safety of synchronous fault-tolerant algorithms by bounded model checking. In: TACAS. pp. 357–374. Springer (2019)
33. Tran, T.H., Konnov, I., Widder, J.: FORTE2021-FD, URL: https://github.com/banhday/forte2021-fd, accessed: January 2021
34. Tran, T.H., Konnov, I., Widder, J.: Cutoffs for symmetric point-to-point distributed algorithms. In: NETYS. pp. 329–346. Springer (2020)
35. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: BFT consensus with linearity and responsiveness. In: PODC. pp. 347–356 (2019)
36. Yu, Y., Manolios, P., Lamport, L.: Model checking TLA$^+$ specifications. In: Correct Hardware Design and Verification Methods, pp. 54–66. Springer (1999)

APPENDIX

A Cutoffs of symmetric point-to-point algorithms under partial synchrony

In [34], Thanh-Hai et al. introduced a class of symmetric point-to-point distributed algorithms, and the failure detector [10] is one example of this class. Thanh-Hai et al. also proved two following cutoffs for this class under either synchrony or asynchrony:

1. Let i be an index, and $\omega_{\{i\}}$ be an LTL\X (the stuttering-insensitive linear temporal logic) formula in which every predicate takes one of the forms: $P_1(i)$ or $P_2(i, i)$. Properties of the form $\bigwedge_{i \, \in \, 1..N} \omega_{\{i\}}$ have a cutoff of 1.
2. Let i and j be different indexes, and $\psi_{\{i,j\}}$ be an LTL\X formula in which every predicate takes one of the forms: $Q_1(i)$, or $Q_2(j)$, or $Q_3(i, j)$, or $Q_4(j, i)$. Properties of the form $\bigwedge_{i,j \, \in \, 1..N}^{i \neq j} \psi_{\{i,j\}}$ have a cutoff of 2.

In this section, we extend their formalization of the global transition system, and their cutoffs on a number of processes for partial synchrony in case of unknown bounds Δ and Φ. It is straightforward to adapt our approach to other models of partial synchrony in [14,10].

Time parameters in partial synchrony only reduce the execution space compared to asynchrony. Hence, we can formalize the system behaviors under partialy synchrony by extending the formalization of the system behaviors under asynchrony in [34] with the notion of time, message ages, time constraints, and admissible sequences of configurations under partial synchrony.

The notion of time, message ages, time constraints, and admissible sequences of configurations under partial synchrony are defined as follows. From now on, we denote $\mathcal{A}$ a symmetric point-to-point algorithm with unknown bounds Δ and Φ. Moreover, we denote $\mathcal{G}_N$ a global transition system of an instance of $\mathcal{A}$ with N processes that is formalized in [34]. We denote $1..N$ a set of process indexes.

Time. Let $\pi = \kappa_0 \kappa_1 \dots$ be an admissible sequence of global configurations in $\mathcal{G}_N$ [1]. We say that the configuration κ_0 is at time 0, and that four configurations $\kappa_{4k-3}, \dots, \kappa_{4k}$ are at time k for every $k > 0$.

Recall that in [34], a global transition contains a sequence of four sub-rounds: *Schedule, Send, Receive,* and *Computation.* In an admissible sequence $\pi = \kappa_0 \kappa_1 \dots$ of global configurations in $\mathcal{G}_N$, for every $k > 0$, every sub-sequence of four configurations $\kappa_{4k-3} \dots \kappa_{4k}$ presents one global transition. Configuration κ_{4k-3} is in sub-round Schedule, and configuration κ_{4k} is in sub-round Computation for every $k > 0$.

Message ages. Now we discuss the formalization of message ages. For every sent message m, the global system packages it with a tag containing its current age, i.e., (m, age_m). When message m has been added to the message buffer in sub-round Send, its age is 0. In our formalization, message ages are increased by 1 in sub-round Schedule. Formally, for every time $k \geq 0$, for every process $s, r \in 1..N$, the following constraints hold:

[1] The definition of an admissible sequence of configurations is provided in [34].

1. For every message (m, age_m) in $buf(\kappa_{4k}, s, r)$, there exists $(m', age_{m'})$ in $buf(\kappa_{4k+1}, s, r)$ such that $m = m'$ and $age_m = age_{m'} + 1$.
2. For every message $(m', age_{m'})$ in $buf(\kappa_{4k+1}, s, r)$, there exists (m, age_m) in $buf(\kappa_{4k}, s, r)$ such that $m = m'$ and $age_m = age_{m'} + 1$.

The first constraint ensures that every in-transit message age will be added one time-unit. The second constraint ensures that no new messages will be added in $buf(\kappa_{4k+1}, s, r)$.

Moreover, the age of an in-transit message is unchanged in other sub-rounds. Formally, for every time $k > 0$, for every $0 \leq \ell \leq 3$, for every pair of processes $s, r \in 1..N$, for every message (m, age_m) in $buf(\kappa_{4k-\ell}, s, r)$, there exists $(m', age_{m'})$ in $buf(\kappa_{4k-3}, s, r)$ such that $m = m'$ and $age_m = age_{m'}$.

Partial synchrony constraints. With the above definitions, Constraints 1 and 2 can be formalized as follows, respectively:

(a) For every process $r \in 1..N$, for every time $k > 0$, if $Enable(\kappa_{4k-1}, r, Loc_{rcv}) = \top$, then for every process $s \in 1..N$, there exists no message (m, age_m) in $buf(\kappa_{4k-1}, s, r)$ such that $age_m \geq \Delta$.

(b) For every process $i \in 1..N$, for every time interval $[k, k + \Phi]$, if we have that $pc(lstate(\kappa_\ell, i)) \neq \ell_{crash}$ for every configuration index $4k - 3 \leq \ell \leq 4(k + \Phi)$, then there exists a configuration index $4k - 3 \leq \ell_0 \leq 4(k + \Phi)$ such that $Enable(\kappa_{\ell_0}, i, loc) = \top$ where loc is one of locations Loc_{snd}, Loc_{rcv}, or Loc_{comp}.

Predicate $Enable(\kappa_k, r, loc)$ refers to whether process p takes a step in configuration κ_k for every $k \geq 0$.

Admissible sequences of configurations under partial synchrony. Let $\pi = \kappa_0\kappa_1 \ldots$ be an admissible sequence of global configurations in $\mathcal{G}_N$. We say that π is under partial synchrony if Constraints 1 and 2 hold in π.

Cutoffs. Now we discuss the cutoffs for symmetric point-to-point algorithms with unknown bounds Δ and Φ that are presented in Theorems 1 and 2.

Theorem 1. *Let $\mathcal{A}$ be a symmetric point–to–point algorithm under partial synchrony with unknown bounds Δ and Φ. Let $\mathcal{G}_1$ and $\mathcal{G}_N$ be instances of $\mathcal{A}$ with 1 and N processes respectively for some $N \geq 1$. Let $Path_1$ and $Path_N$ be sets of all admissible sequences of configurations in $\mathcal{G}_1$ and in $\mathcal{G}_N$ under partial synchrony, respectively. Let $\omega_{\{i\}}$ be a LTL\X formula in which every predicate takes one of the forms: $P_1(i)$ or $P_2(i, i)$ where i is an index in $1..N$. It follows that*

$$\left(\forall \pi_N \in Path_N : \mathcal{G}_N, \pi_N \models \bigwedge_{i \in 1..N} \omega_{\{i\}}\right) \Leftrightarrow \left(\forall \pi_1 \in Path_1 : \mathcal{G}_1, \pi_1 \models \omega_{\{1\}}\right)$$

Theorem 2. *Let $\mathcal{A}$ be a symmetric point–to–point algorithm under partial synchrony with unknown bounds Δ and Φ. Let $\mathcal{G}_2$ and $\mathcal{G}_N$ be instances of $\mathcal{A}$ with 2 and N processes respectively for some $N \geq 2$. Let $Path_2$ and $Path_N$ be sets of all admissible sequences of configurations in $\mathcal{G}_2$ and in $\mathcal{G}_N$ under partial synchrony, respectively. Let $\psi_{\{i,j\}}$ be an LTL\X formula in which every predicate takes one of the forms: $Q_1(i)$, or $Q_2(j)$, or $Q_3(i, j)$, or $Q_4(j, i)$ where i and j are different indexes in $1..N$. It follows that:*

$$\left(\forall \pi_N \in Path_N : \mathcal{G}_N, \pi_N \models \bigwedge_{i,j \in 1..N}^{i \neq j} \psi_{\{i,j\}}\right) \Leftrightarrow \left(\forall \pi_2 \in Path_2 : \mathcal{G}_2, \pi_2 \models \psi_{\{1,2\}}\right)$$

Since the proofs of these theorems are similar, we here focus on only Theorem 2. The proof of Theorem 2 follows the approach in [15,34], and is based on the following observations:

- The global transition system and the desired property are symmetric [34].
- Let $\mathcal{G}_2$ and $\mathcal{G}_N$ be two instances of a symmetric point-to-point algorithm with 2 and N processes, respectively. By [34], two instances $\mathcal{G}_2$ and $\mathcal{G}_N$ are trace equivalent under a set of predicates in the desired property.
- We will now discuss that the constraints maintain partial synchrony. Let π_N be an execution in $\mathcal{G}_N$. By applying the index projection to π_N, we obtain an execution π_2 in $\mathcal{G}_2$ [34]. Intuitively, the index projection throws away processes $3..N$ as well as their corresponding messages and buffers. Moreover, for every $i, j \in \{1, 2\}$, the index projection preserves (i) when process i takes a step, and (ii) what action process i takes at time $t \geq 0$, and (iii) messages between process i and process j. For example, Figure 1 demonstrates an execution in $\mathcal{G}_2$ that is constructed based on a given execution in $\mathcal{G}_3$ with the index projection. The index projection is defined in [34]. If partial synchrony constraints hold on π_N, these constraints also hold on π_2. This result is proved in Lemma 1.
- Let π_2 be an execution in $\mathcal{G}_2$. We construct an execution π_N in $\mathcal{G}_N$ based on π_2 such that all processes $3..N$ crash from the beginning, and π_2 is an index projection of π_N [34]. For instance, Figure 2 demonstrates an execution in $\mathcal{G}_3$ that is constructed based on one in $\mathcal{G}_2$. If partial synchrony constraints hold on π_2, these constraints also hold on π_N. This result is proved in Lemma 2.

Lemma 1. *Let $\mathcal{A}$ be a symmetric point–to–point algorithm under partial synchrony with unknown bounds Δ and Φ. Let $\mathcal{G}_2$ and $\mathcal{G}_N$ be instances of $\mathcal{A}$ with 2 and N processes, respectively, for some $N \geq 2$. Let $Path_2$ and $Path_N$ be sets of all admissible sequences of configurations in $\mathcal{G}_2$ and in $\mathcal{G}_N$ under partial synchrony, respectively. Let $\pi^N = \kappa_0^N \kappa_1^N \dots$ be an admissible sequences of configurations in $\mathcal{G}_N$. Let $\pi^2 = \kappa_0^2 \kappa_1^2 \dots$ be a sequence of configurations in $\mathcal{G}_2$ such that κ_k^2 be an index projection of κ_k^N on indexes $\{1, 2\}$ for every $k \geq 0$. It follows that*
(a) Constraint 1 holds on π^2.
(b) Constraint 2 holds on π^2.

Proof. Recall that the index projection is defined on Page 13 [34]. In the following, we denote p^2 and p^N two processes such that they have the same index, and p^2 is a process in $\mathcal{G}_2$, and p^N is a process in $\mathcal{G}_N$. We prove Lemma 1 by contradiction.

(a) Assume that Constraint 1 does not hold on π^2. Hence, there exist a time $\ell > 0$, and two processes $s^2, r^2 \in 1..2$ in $\mathcal{G}_2$ such that after r^2 executes Receive at a time ℓ, there exists an old message in a message buffer from process s^2 to process r^2. By the definition of the index projection, for every $k \geq 0$, we have that:

- Let $p^N, q^N \in \{1, 2\}$ be two processes in $\mathcal{G}_N$, and p^2, q^2 be corresponding processes in $\mathcal{G}_2$. For every $k \geq 0$, two message buffers from process p^2 to process q^2 in κ_k^2, and from process p^N to process q^N in κ_k^N are the same.

– Let $p^N \in \{1, 2\}$ be a process in $\mathcal{G}_N$, and p^2 be a corresponding process in $\mathcal{G}_2$. For every $k \geq 0$, process p^2 takes an action *act* in configuration κ^2_k if and only if process p^N takes the same action in configuration κ^N_k.

It implies that process r^N in $\mathcal{G}_N$ also executes Receive at a time ℓ, and there exists an old message in a buffer from process s^N to process r^N. Contradiction.

(b) By applying similar arguments in case (a). ■

Lemma 2. *Let $\mathcal{A}$ be a symmetric point–to–point algorithm under partial synchrony with unknown bounds Δ and Φ. Let $\mathcal{G}_2$ and $\mathcal{G}_N$ be instances of $\mathcal{A}$ with 2 and N processes, respectively, for some $N \geq 2$. Let $Path_2$ and $Path_N$ be sets of all admissible sequences of configurations in $\mathcal{G}_2$ and in $\mathcal{G}_N$ under partial synchrony, respectively. Let $\pi^2 = \kappa^2_0\kappa^2_1 \ldots$ be an admissible sequences of configurations in $\mathcal{G}_2$. Let $\pi^N = \kappa^N_0\kappa^N_1 \ldots$ be a sequence of configurations in $\mathcal{G}_N$ such that (i) every process $p \in 3..N$ crashes from the beginning, and (ii) κ^2_k be an index projection of κ^N_k on indexes $\{1, 2\}$ for every $k \geq 0$. It follows that*

(a) Constraint 1 holds on π^N.

(b) Constraint 2 holds on π^N.

Proof. By applying similar arguments in the proof of Lemma 1, and the facts that every process $p \in 3..N$ crashes from the beginning, and that κ^2_k be an index projection of κ^N_k on indexes $\{1, 2\}$ for every $k \geq 0$. ■

B TLA$^+$specification: the failure detector

In this section, we present the TLA$^+$ specification of the failure detector with our inductive invariant for the safety property Strong accuracy.

```
──────────────────── MODULE sa_iinv_pred ────────────────────

EXTENDS Naturals, TLC

CONSTANTS Phi, Delta                 The bounds
ASSUME 1 ≤ Phi

ASSUME Delta ≥ 0

TODefault ≜ 6 ∗ Phi + Delta          Default value of timeout
ASSUME TODefault ≥ 6 ∗ Phi + Delta

Action  ≜ {"SSched", "RSched", "IncMsgAge", "SSnd", "RNoSnd", "RRcv", "RComp"}
MsgAge ≜ 0 . . Delta


VARIABLES suspected,          whether the receiver suspects the sender
          waitingTime,        how long the receiver has not heard from the sender
          timeout,            how long the receiver can wait for the sender
          sPC,                pc of the sender
          rPC,                pc of the receiver
          existsMsgOfAge,     the communication channel that is encoded with this predicate
          sTimer,             how long the sender has not taken a step
          rTimer,             how long the sender has not taken a step
          ePC                 pc of the environment

vars ≜ ⟨existsMsgOfAge, sTimer, rTimer, ePC,
         suspected, waitingTime, timeout, sPC, rPC⟩

 The initialization
Schedule_Init ≜
   ∧ sTimer = 0
   ∧ rTimer = 0
   ∧ ePC = "SSched"

Proc_Init ≜
   ∧ sPC = "SSnd"
   ∧ rPC = "RComp"
   ∧ existsMsgOfAge = [i ∈ MsgAge ↦ FALSE]
   ∧ suspected = FALSE
   ∧ waitingTime = 0
   ∧ timeout = TODefault

Init ≜ Proc_Init ∧ Schedule_Init
```

```
$SSched \triangleq$ schedule the sender
  $\land ePC = \text{"SSched"}$
  $\land ePC' = \text{"RSched"}$
  $\land \lor \land sTimer \geq 2$
       $\land sTimer' = 0$
       $\land$ UNCHANGED $sPC$
    $\lor \land sTimer < 3 * Phi - 1$
       $\land sTimer' = sTimer + 1$
       $\land$ UNCHANGED $sPC$
  $\land$ UNCHANGED $\langle existsMsgOfAge, rTimer, rPC, suspected, waitingTime, timeout \rangle$

$RSched \triangleq$ schedule the receiver
  $\land ePC = \text{"RSched"}$
  $\land ePC' = \text{"IncMsgAge"}$
  $\land \lor \land rTimer' = 0$
       $\land \lor \land rPC = \text{"RNoSnd"}$
               $\land rPC' = \text{"RRcv"}$
            $\lor \land rPC = \text{"RRcv"}$
               $\land rPC' = \text{"RComp"}$
            $\lor \land rPC = \text{"RComp"}$
               $\land rPC' = \text{"RNoSnd"}$
    $\lor \land rTimer < Phi - 1$
       $\land rTimer' = rTimer + 1$
       $\land$ UNCHANGED $rPC$
  $\land$ UNCHANGED $\langle existsMsgOfAge, sTimer, sPC, suspected, waitingTime, timeout \rangle$

$IncMsgAge \triangleq$ increase the age of in-transit messages
  $\land ePC = \text{"IncMsgAge"}$
  $\land ePC' = \text{"SSnd"}$
  $\land$ IF $Delta = 0$
      THEN UNCHANGED $existsMsgOfAge$
      ELSE $\land existsMsgOfAge' \in [MsgAge \rightarrow \text{BOOLEAN}]$
           $\land existsMsgOfAge'[Delta] = (existsMsgOfAge[Delta] \lor existsMsgOfAge[Delta - 1])$
           $\land \neg existsMsgOfAge'[0]$
           $\land \forall k \in 0 \,..\, (Delta - 2) : existsMsgOfAge'[k + 1] = existsMsgOfAge[k]$
  $\land$ UNCHANGED $\langle sTimer, rTimer,$
          $suspected, waitingTime, timeout, sPC, rPC \rangle$

$SSnd \triangleq$ send a new message
  $\land ePC = \text{"SSnd"}$
  $\land ePC' = \text{"RNoSnd"}$
  $\land$ IF $(sTimer = 0 \land sPC = \text{"SSnd"})$ whether the sender is active in the sub-round Send
       THEN $existsMsgOfAge' = [existsMsgOfAge$ EXCEPT $![0] = \text{TRUE}]$
       ELSE UNCHANGED $existsMsgOfAge$
  $\land$ UNCHANGED $\langle sTimer, rTimer, sPC, rPC, suspected, waitingTime, timeout \rangle$

$RNoSnd \triangleq$   the receiver in sub-round Send
  $\land$ $ePC =$ "RNoSnd"
  $\land$ $ePC' =$ "RRcv"
  $\land$ IF $(rTimer = 0 \land rPC =$ "RNoSnd"$)$   is the receiver active in the sub-round Send?
     THEN $\land$ IF $waitingTime < timeout$
               THEN $waitingTime' = waitingTime + 1$
               ELSE UNCHANGED $waitingTime$
          $\land$ UNCHANGED $\langle existsMsgOfAge, sTimer, rTimer, sPC, rPC,$
                          $suspected, timeout \rangle$
     ELSE UNCHANGED $\langle existsMsgOfAge, sTimer, rTimer,$
          $suspected, waitingTime, timeout, sPC, rPC \rangle$

$MsgDeliver \triangleq$   at least one in-transit message will be delivered
  $\land$ $\exists k \in MsgAge : existsMsgOfAge[k]$
  $\land$ $existsMsgOfAge' \in [MsgAge \rightarrow$ BOOLEAN $]$
  $\land$ $\neg existsMsgOfAge'[Delta]$
  $\land$ $\exists k \in MsgAge : existsMsgOfAge'[k] \neq existsMsgOfAge[k]$
  $\land$ $\forall k \in MsgAge : existsMsgOfAge'[k] \implies existsMsgOfAge[k]$
  $\land$ $waitingTime' = 0$
  $\land$ IF $suspected$                          did the receiver suspect the sender?
     THEN $\land$ $suspected' =$ FALSE            trust the sender again
          $\land$ $timeout' = timeout + 1$          increase the time out
     ELSE UNCHANGED $\langle suspected, timeout \rangle$
  $\land$ UNCHANGED $\langle sTimer, rTimer, sPC, rPC \rangle$

$NoMsgDeliver \triangleq$   no messages are delivered
  $\land$ $\neg existsMsgOfAge[Delta]$
  $\land$ UNCHANGED $existsMsgOfAge$
  $\land$ IF $waitingTime < timeout$
     THEN $waitingTime' = waitingTime + 1$
     ELSE UNCHANGED $waitingTime$
  $\land$ UNCHANGED $\langle suspected, timeout, sTimer, rTimer, sPC, rPC \rangle$

$RRcv \triangleq$   the receiver in sub-round Receive
  $\land$  $ePC =$ "RRcv"
  $\land$  $ePC' =$ "RComp"
  $\land$  IF $(rTimer = 0 \land rPC =$ "RRcv"$)$   is the receiver active in the sub-round Receive?
      THEN $\lor$ $MsgDeliver$
           $\lor$ $NoMsgDeliver$
      ELSE UNCHANGED $\langle existsMsgOfAge, sTimer, rTimer,$
           $suspected, waitingTime, timeout, sPC, rPC \rangle$
```

```
$RComp \triangleq$   the receiver in sub-round Local Computation
  $\land\ ePC = \text{“RComp”}$
  $\land\ ePC' = \text{“SSched”}$
  $\land$ IF $rTimer = 0 \land rPC = \text{“RComp”}$
      THEN $\land$ IF $waitingTime < timeout$
              THEN $\land\ waitingTime' = waitingTime + 1$      increase the waiting time
                   $\land$ UNCHANGED $suspected$                 still trust the sender
              ELSE $\land$ IF $\neg suspected$
                        THEN $suspected' =$ TRUE
                        ELSE UNCHANGED $suspected$
                   $\land$ UNCHANGED $waitingTime$
           $\land$ UNCHANGED $\langle existsMsgOfAge, sTimer, rTimer, sPC, rPC, timeout\rangle$
      ELSE UNCHANGED $\langle existsMsgOfAge, sTimer, rTimer, sPC, rPC,$
                $waitingTime, timeout, suspected\rangle$

$Next \triangleq$
  $\lor\ SSched$
  $\lor\ RSched$
  $\lor\ IncMsgAge$
  $\lor\ SSnd$
  $\lor\ RNoSnd$
  $\lor\ RRcv$
  $\lor\ RComp$

$Spec \triangleq Init \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(Next)$

$TypeOK \triangleq$
  $\land\ existsMsgOfAge \in [MsgAge \rightarrow$ BOOLEAN $]$
  $\land\ sTimer \in 0 \,..\, (3 * Phi - 1)$
  $\land\ rTimer \in 0 \,..\, (Phi - 1)$
  $\land\ ePC \in \{\text{“SSched”}, \text{“RSched”}, \text{“IncMsgAge”}, \text{“SSnd”},$
              $\text{“RNoSnd”}, \text{“RRcv”}, \text{“RComp”}\}$
  $\land\ suspected \in$ BOOLEAN
  $\land\ timeout = 6 * Phi + Delta$
  $\land\ waitingTime \in 0 \,..\, (timeout - 1)$
  $\land\ sPC \in \{\text{“SSnd”}\}$
  $\land\ rPC \in \{\text{“RNoSnd”}, \text{“RRcv”}, \text{“RComp”}\}$

$StrongAccuracy \triangleq \neg suspected$

$Constraint1 \triangleq$
  $(\land (\forall k \in MsgAge : \neg existsMsgOfAge[k])$
   $\land\ waitingTime \geq 0$
   $\land\ waitingTime < 3 * Phi)$
```

$\qquad\implies sTimer \geq waitingTime$

$Constraint2 \triangleq$
$\quad \land (3 * Phi \leq waitingTime \land waitingTime < 3 * Phi + Delta)$
$\qquad\qquad \implies (\exists\, k \in MsgAge : waitingTime \leq 3 * Phi + k \land existsMsgOfAge[k])$
$\quad \land waitingTime \geq 3 * Phi + Delta \implies existsMsgOfAge[Delta]$

$Constraint3 \triangleq$
$\quad \land (waitingTime \geq 5 * Phi + Delta$
$\qquad\qquad \implies (rTimer \geq waitingTime - 5 * Phi - Delta \land rPC = \text{“RNoSnd”}))$
$\quad \land ((waitingTime \geq 4 * Phi + Delta \land waitingTime < 5 * Phi + Delta)$
$\qquad\qquad \implies (\land (rPC = \text{“RNoSnd”} \lor rPC = \text{“RComp”})$
$\qquad\qquad\qquad \land (rPC = \text{“RComp”} \implies rTimer \geq waitingTime - 4 * Phi - Delta)))$
$\quad \land ((waitingTime \geq 3 * Phi + Delta \land waitingTime < 4 * Phi + Delta)$
$\qquad\qquad \implies (rPC = \text{“RRcv”} \implies rTimer > waitingTime - 3 * Phi - Delta))$

$ConstraintsInIndInv\_BigStep \triangleq$
$\quad \land TypeOK$
$\quad \land StrongAccuracy$
$\quad \land Constraint1$
$\quad \land Constraint2$
$\quad \land Constraint3$

$IndInv\_BigTODefault \triangleq$
$\quad \lor ePC \neq \text{“SSched”}$
$\quad \lor ConstraintsInIndInv\_BigStep$

$Init\_IndInv\_BigTODefault \triangleq$
$\quad \land ePC = \text{“SSched”}$
$\quad \land ConstraintsInIndInv\_BigStep$

C An inductive invariant with Strong Accuracy

In this section, we explain how we construct an inductive invariant with Strong Accuracy. For convenience, we write invariants in first-order logic, instead of the Ivy syntax [30].

C.1 Type invariants

The following formulas are type invariants.

$$0 \leq \mathit{\Delta} \wedge 1 \leq \mathit{\Phi} \tag{15}$$
$$\mathsf{sPC} = \mathsf{sSnd} \tag{16}$$
$$\mathsf{rPC} = \mathsf{rNoSnd} \vee \mathsf{rPC} = \mathsf{rRcv} \vee \mathsf{rPC} = \mathsf{rComp} \tag{17}$$
$$0 \leq \mathsf{sTimer} \wedge \mathsf{sTimer} < 3 \times \mathit{\Phi} \tag{18}$$
$$0 \leq \mathsf{rTimer} \wedge \mathsf{rTimer} < \mathit{\Phi} \tag{19}$$
$$\mathsf{timeout} \geq 6 \times \mathit{\Phi} + \mathit{\Delta} \tag{20}$$
$$0 \leq \mathsf{waitingTime} \tag{21}$$
$$\forall x \in \mathbb{Z} \,.\, (x < 0 \vee \mathit{\Delta} < x) \Rightarrow \neg\mathsf{existsMsgOfAge}(x) \tag{22}$$
$$\forall x \in \mathbb{Z} \,.\, \mathsf{tmpExistsMsgOfAge}(x) = \mathsf{existsMsgOfAge}(x) \tag{23}$$
$$\mathsf{constant} \geq 6 \times \mathit{\Phi} + \mathit{\Delta} \Rightarrow \mathsf{constant} = \mathsf{timeout} \tag{24}$$

Initially, we set $\mathsf{constant} = \mathsf{timeout}$. By Invariants 20 and 24, we have that $\mathsf{timeout}$ is unchanged.

C.2 Strong Accuracy

Under our assumptions and encoding, the safety property Strong Accuracy can be rewritten as the following formula.

$$\neg\mathsf{suspected} \wedge \mathsf{waitingTime} < \mathsf{timeout} \tag{25}$$

C.3 waitingTime, existsMsgOfAge and sTimer

The following invariant refers to the fact that if $\mathsf{waitingTime} \geq 3 * \mathit{\Phi}$, then there exists at least one in-transit message.

$$\begin{aligned}&\big((\forall x \in \mathbb{Z} \,.\, (0 \leq x \leq \mathit{\Delta}) \Rightarrow \neg\mathsf{existsMsgOfAge}(x))\\ &\quad \wedge \mathsf{waitingTime} > 0\\ &\quad \wedge \mathsf{waitingTime} < 3 \times \mathit{\Phi}\big)\\ &\qquad\qquad \Rightarrow \mathsf{sTimer} \geq \mathsf{waitingTime}\end{aligned} \tag{26}$$

Explanation. From now on, we denote $\mathsf{eval}(\mathit{obj}, t)$ the value of obj at time t where obj is either a variable or a predicate, e.g., $\mathsf{eval}(\mathsf{waitingTime}, 2)$ refers to the value of $\mathsf{waitingTime}$ at time 2.

Let t_0 be the last time point where $\mathsf{eval}(\mathsf{waitingTime}, t_0) = 0$ (a). Let t be the current time point. We have:

(b) $\forall x \in \mathbb{Z} \,.\, (0 \leq x \leq \Delta) \Rightarrow \neg\mathsf{eval}(\mathsf{existsMsgOfAge}(x), t)$
(c) $\mathsf{eval}(\mathsf{waitingTime}, t) > 0$
(d) $\mathsf{eval}(\mathsf{waitingTime}, t) < 3 * \Phi$

Moreover, we have that

$$\begin{aligned}\forall t' :& t_0 \leq t' \leq t \Rightarrow \\ &\big(\mathsf{eval}(\mathsf{waitingTime}, t') \geq 0 \\ &\quad \wedge (\forall x \in \mathbb{Z} \,.\, (0 \leq x \leq \Delta) \Rightarrow \neg\mathsf{eval}(\mathsf{existsMsgOfAge}(x), t'))\big)\end{aligned}$$

We know that $t - t_0 \geq \mathsf{eval}(\mathsf{waitingTime}, t)$ since (i) the receiver might be inactive in some time point between t_0 and t, and (ii) waitingTime is measured with the local clock of the receiver, and (iii) t and t_0 are measured with the global clock that is not slower than any local clock.
By (a) and (c), the communication channel is empty in the time interval $[t_0, t]$. Hence, the sender has not sent any message from t_0. It implies that sTimer has not been reset to 0 in the time interval $[t_0, t]$. Since $\mathsf{eval}(\mathsf{sTimer}, t_0) \geq 0$ and $\mathsf{eval}(\mathsf{sTimer}, t) = \mathsf{eval}(\mathsf{sTimer}, t_0) + t - t_0$, we have $\mathsf{eval}(\mathsf{sTimer}, t) \geq t - t_0$. Since $\mathsf{eval}(\mathsf{waitingTime}, t_0) = 0$ and waitingTime is measured with the local clock of the receiver, we have that $t - t_0 \geq \mathsf{eval}(\mathsf{waitingTime}, t)$. Hence, it implies that $\mathsf{eval}(\mathsf{sTimer}, t) \geq \mathsf{eval}(\mathsf{waitingTime}, t)$.

C.4 waitingTime and the age of in-transit messages

The following invariant describes the relationship between waitingTime and ages of in-transit messages.

$$\begin{aligned}\big(3 \times \Phi \leq \mathsf{waitingTime} \wedge \mathsf{waitingTime} < 3 \times \Phi + \Delta\big) \\ \Rightarrow \big(\exists x \in \mathbb{Z} \,.\, 0 \leq x \leq \Delta \wedge \mathsf{waitingTime} \leq 3 \times \Phi + x \\ \wedge\ \mathsf{existsMsgOfAge}(x)\big)\end{aligned} \tag{27}$$

Explanation. Let t_0 be the last time point where $\mathsf{eval}(\mathsf{waitingTime}, t_0) = 3 * \Phi$. By Invariant C.4, we have that there exists a number k such that $0 \leq k \leq \Delta$ and $\mathsf{eval}(\mathsf{existsMsgOfAge}(k), t_0) = \mathtt{true}$ at time t_0.
Now let t be the current time point. We have

$$3 \times \Phi \leq \mathsf{eval}(\mathsf{waitingTime}, t) < 3 \times \Phi + \Delta$$

Because of the premise in Invariant 27, we know that the receiver has not received any message from the sender in the time interval $[t_0, t]$. Moreover, message ages are increased by 1 after every global transition. Hence, one of predicates $\mathsf{eval}(\mathsf{existsMsgOfAge}(k + t - t_0), t)$ and $\mathsf{eval}(\mathsf{existsMsgOfAge}(\Delta), t)$ must be true.

C.5 waitingTime and the existance of old messages

By Invariant C.4, the following invariant holds. This invariant implies that if $\mathsf{waitingTime} \geq 3 \times \Phi + \Delta$, then there exists at least one old message.

$$\mathsf{waitingTime} \geq 3 \times \Phi + \Delta \Rightarrow \mathsf{existsMsgOfAge}(\Delta) \tag{28}$$

C.6 waitingTime is bounded by $6 \times \Phi + \Delta$

Invariant 29 implies that $\mathsf{waitingTime}$ never reaches the guard $6 \times \Phi + \Delta$.

$$\mathsf{waitingTime} < 6 \times \Phi + \Delta \tag{29}$$

Three following Invariants 30–32 support Invariant 29. Moreover, Invariants 30–32 support an observation that (i) if $\mathsf{waitingTime}$ is $6 \times \Phi + \Delta - 1$ at a time t_1 and (ii) if the receiver is active at a time t_2 after a time t_1 and (iii) if the receiver is always inactive in the time interval $[t_1, t_2 - 1]$, then the receiver must execute an instruction Receive at a time t_2.

$$\Big((\mathsf{waitingTime} \geq 5 \times \Phi + \Delta) \Rightarrow \\ (rTimer \geq \mathsf{waitingTime} - 5 \times \Phi - \Delta \wedge \mathsf{rPC} = \mathsf{rNoSnd})\Big) \tag{30}$$

$$\Big((4 \times \Phi + \Delta \leq \mathsf{waitingTime} < 5 \times \Phi + \Delta) \Rightarrow \\ (\quad (\mathsf{rPC} = \mathsf{rNoSnd} \vee \mathsf{rPC} = \mathsf{rComp}) \\ \wedge (\mathsf{rPC} = \mathsf{rComp} \Rightarrow \mathsf{rTimer} \geq \mathsf{waitingTime} - 4 \times \Phi - \Delta))\Big) \tag{31}$$

$$\Big((3 \times \Phi + \Delta \leq \mathsf{waitingTime} < 4 \times \Phi + \Delta) \Rightarrow \\ (\mathsf{rPC} = \mathsf{rRcv} \Rightarrow \mathsf{rTimer} > \mathsf{waitingTime} - 3 \times \Phi - \Delta)\Big) \tag{32}$$

– *Invariant 30.* Let t be the current time point when we have

$$\mathsf{eval}(\mathsf{waitingTime}, t) \geq 5 \times \Phi + \Delta$$

Let t_0 be the last time point when $\mathsf{eval}(\mathsf{waitingTime}, t_0) = 3 \times \Phi + \Delta$. By Invariant C.5, we have that $\mathsf{eval}(\mathsf{existsMsgOfAge}(\Delta), t_0) = \mathtt{true}$. Moreover, we have $t - t_0 \geq 2 * \Phi$. Hence, in the time interval $[t_0, t]$ the receiver has to be active at least twice. Due to $\mathsf{eval}(\mathsf{waitingTime}, t) \geq 5 \times \Phi + \Delta$, the receiver has not executed any instruction Receive in the time interval $[t_0, t]$. It implies that the receiver executed an instruction Compute, and then an instruction No Send. Hence, $\mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rNoSnd}$. Moreover, we know that the last time when the receiver is active is by $5 \times \Phi + \Delta$. Since rTimer is measured with the global clock, and $\mathsf{waitingTime}$ is measured with the local clock of the receiver, we have that

$$\mathsf{eval}(\mathsf{rTimer}, t) \geq \mathsf{eval}(\mathsf{waitingTime}, t) - 5 \times \Phi - \Delta$$

– *Invariant 31* Let t be the current time point when we have

$$4 \times \Phi + \Delta \leq \mathsf{eval}(\mathsf{waitingTime}, t) < 5 \times \Phi + \Delta$$

Let t_0 be the last time point when $\mathsf{eval}(\mathsf{waitingTime}, t_0) = 3 \times \Phi + \Delta$. By similar arguments in the cases of $\mathsf{waitingTime} \geq 5 \times \Phi + \Delta$, we know that the receiver did not execute any instruction Receive in the time interval $[t_0, t]$. Moreover, in the time interval $[t_0, t]$, the receiver has to be active at least once. Hence, we have that $\mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rNoSnd} \vee \mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rComp}$ at time t.
Now assume $\mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rComp}$. Let t_R be the last time point when the receiver executed Receive. Since $\mathsf{eval}(\mathsf{waitingTime}, t_0) = 3 \times \Phi + \Delta$ and $\mathsf{eval}(\mathsf{waitingTime}, t) \geq 4 \times \Phi + \Delta$, we know that t_R is before t_0, i.e., $t_R < t_0$. By applying similar arguments in Invariant 26, we have that

$$\mathsf{eval}(\mathsf{rTimer}, t) \geq \mathsf{eval}(\mathsf{waitingTime}, t) - 4 \times \Phi - \Delta$$

– *Invariant 32*. Let t be the current time point when we have

$$3 \times \Phi + \Delta \leq \mathsf{eval}(\mathsf{waitingTime}, t) < 4 \times \Phi + \Delta$$

Let t_0 be the last time point where $\mathsf{eval}(\mathsf{waitingTime}, t_0) = 3 \times \Phi + \Delta$. By similar arguments in the cases of $\mathsf{waitingTime} \geq 5 \times \Phi + \Delta$, we know that the receiver did not execute any instruction Receive in the time interval $[t_0, t]$. Let t_R be the last time point when the receiver executed an instruction Receive. By similar arguments in case of $\mathsf{waitingTime} \geq 4 \times \Phi + \Delta \wedge \mathsf{waitingTime} < 5 \times \Phi + \Delta$, we know that t_R is before t_0, i.e., $t_R < t_0$. Now assume that $\mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rRcv}$. By applying similar arguments in Invariant 26, we have

$$\mathsf{eval}(\mathsf{rTimer}, t) \geq \mathsf{eval}(\mathsf{waitingTime}, t) - 3 \times \Phi - \Delta$$

D An inductive invariant with Eventually Strong Accuracy

D.1 Type invariants

The type invariants are similar to those in an inductive invariant with Strong Accuracy, except that Invariant 20 is replaced with Invariant 33.

$$1 \leq \mathsf{timeout} \tag{33}$$

D.2 Eventually Strong Accuracy

Under our assumptions, encoding and reduction, we can rewrite the property Eventually Strong Accuracy with the following formula

$$\begin{aligned} &\mathsf{timeout} \geq 6 \times \Phi + \Delta \Rightarrow \neg\mathsf{suspected} \land \mathsf{waitingTime} < \mathsf{timeout} && (34)\\ \land\ &\mathsf{constant} \geq 6 \times \Phi + \Delta \Rightarrow \mathsf{timeout} = \mathsf{constant} && (35)\\ \land\ &\mathsf{constant} < 6 \times \Phi + \Delta \Rightarrow \mathsf{timeout} \leq 6 \times \Phi + \Delta && (36)\end{aligned}$$

Formula 34 refers to that if $\mathsf{timeout} < 6 \times \Phi + \Delta$, then the receiver might have a wrong suspicion of correctness of the sender. Whenever $\mathsf{timeout}$ is greater than or equals $6 \times \Phi + \Delta$, the receiver trusts the sender forever, and $\mathsf{waitingTime}$ is always less than $\mathsf{timeout}$.
Recall that in the initial step, we have $\mathsf{constant} = \mathsf{timeout}$. Formula 35 refers to that $\mathsf{timeout}$ is unchanged if it is greater than or equals $6 \times \Phi + \Delta$. Otherwise, $\mathsf{timeout}$ might increase by 1 after a global transition. That is why Formula 36 holds.

D.3 Other invariants

Our inductive invariant for the liveness property Eventually Strong Accuracy also has invariants in Sections C.3-C.6.

E An inductive invariant with Strong Completeness and arbitrary timeout

Recall that to verify the liveness property Strong Completeness, we introduce two ghost variables: constant to keep track of whether timeout is unchanged, and hLFSC to keep track of how long the sender has crashed. Initially, we have that constant = timeout, and hLFSC = −1, and the sender is correct. Most of the invariants in Section E focus on reachable states, except Invariant 74. Invariant 74 helps Ivy go through the inductive invariant by eliminating some counterexamples that are not reachable states.

E.1 Type invariants

The following constraints are type invariants.

$$0 \leq \mathit{\Delta} \tag{37}$$
$$1 \leq \mathit{\Phi} \tag{38}$$
$$\mathsf{sPC} = \mathsf{sSnd} \lor \mathsf{sPC} = \mathsf{sCrash} \tag{39}$$
$$\mathsf{rPC} = \mathsf{rNoSnd} \lor \mathsf{rPC} = \mathsf{rRcv} \lor \mathsf{rPC} = \mathsf{rComp} \tag{40}$$
$$0 \land \mathsf{sTimer} \land \mathsf{sTimer} < 3 \times \mathit{\Phi} \tag{41}$$
$$0 \land \mathsf{rTimer} \land \mathsf{rTimer} < \mathit{\Phi} \tag{42}$$
$$1 \leq \mathsf{timeout} \tag{43}$$
$$0 \leq \mathsf{waitingTime} \tag{44}$$
$$\forall x \in \mathbb{Z} \,.\, (x < 0 \lor \mathit{\Delta} < x) \Rightarrow \neg \mathsf{existsMsgOfAge}(x) \tag{45}$$
$$\forall x \in \mathbb{Z} \,.\, \mathsf{tmpExistsMsgOfAge}(X) = \mathsf{existsMsgOfAge}(x) \tag{46}$$
$$\mathsf{hLFSC} \geq -1 \tag{47}$$
$$\mathsf{suspected} \Rightarrow \mathsf{waitingTime} \geq \mathsf{timeout} \tag{48}$$
$$\mathsf{sPC} = \mathsf{sCrash} \Leftrightarrow (\mathsf{sTimer} = 0 \land \mathsf{hLFSC} \geq 0) \tag{49}$$
$$\mathsf{sPC} = \mathsf{sSnd} \Leftrightarrow \mathsf{hLFSC} = -1 \tag{50}$$

E.2 Strong completeness

Under our assumptions, encoding, and reduction, the liveness property Strong Completeness can be rewritten as the following:

$$\begin{aligned}
&\big((\mathsf{constant} \geq 6 \times \mathit{\Phi} + \mathit{\Delta} \Rightarrow \mathsf{timeout} = \mathsf{constant}) \\
&\quad \land \ (\mathsf{constant} < 6 \times \mathit{\Phi} + \mathit{\Delta} \Rightarrow \mathsf{timeout} \leq 6 \times \mathit{\Phi} + \mathit{\Delta})\big) && (51) \\
&\land \mathsf{timeout} \geq 6 \times \mathit{\Phi} + \mathit{\Delta} \\
&\qquad \Rightarrow \mathsf{hLFSC} \leq (3 \times \mathit{\Phi} + \mathit{\Delta}) + \mathit{\Phi} \times \mathsf{timeout} + (3 \times \mathit{\Phi}) && (52) \\
&\land \mathsf{hLFSC} \geq (3 \times \mathit{\Phi} + \mathit{\Delta}) + \mathit{\Phi} \times \mathsf{timeout} + (3 \times \mathit{\Phi}) \\
&\qquad \Rightarrow (\mathsf{suspected} \land \mathsf{sPC} = \mathsf{sCrash}) && (53)
\end{aligned}$$

$$\begin{aligned} &\wedge (\mathsf{timeout} \geq 6 \times \mathit{\Phi} + \mathit{\Delta} \wedge \mathsf{waitingTime} = \mathsf{timeout}) \\ &\qquad \Rightarrow (\mathsf{sPC} = \mathsf{sCrash} \wedge \forall x \in \mathbb{Z} \,.\, \neg \mathsf{existsMsgOfAge}(x)) \qquad (54) \\ &\wedge (\mathsf{timeout} \geq 6 \times \mathit{\Phi} + \mathit{\Delta} \wedge \mathsf{waitingTime} > \mathsf{timeout}) \\ &\qquad \Rightarrow (\mathsf{sPC} = \mathsf{sCrash} \wedge \forall x \in \mathbb{Z} \,.\, \neg \mathsf{existsMsgOfAge}(x)) \qquad (55) \end{aligned}$$

Formula 51 refers to that if $\mathsf{constant} \geq 6 \times \mathit{\Phi} + \mathit{\Delta}$, then $\mathsf{timeout}$ is unchanged. Otherwise, $\mathsf{timeout}$ might be changed. If $\mathsf{constant} < 6 \times \mathit{\Phi} + \mathit{\Delta}$, then $\mathsf{timeout}$ might be increased by 1 since the receiver had a wrong suspicion of the correctness of the sender. Hence, Formula 51 implies that $\mathsf{timeout}$ is bounded in every execution. Formula 52 refers to that the maximum value of hLFSC depends on $\mathit{\Phi}, \mathit{\Delta}$ and $\mathsf{timeout}$. Since $\mathsf{timeout}$ is bounded, hLFSC is also bounded. Formula 53 refers to that after variable hLFSC reaches the guard $(3 \times \mathit{\Phi} + \mathit{\Delta}) + \mathit{\Phi} \times \mathsf{timeout} + (3 \times \mathit{\Phi})$, we have that (i) the receiver suspects the sender forever, and (ii) the sender crashed. Formulas 54 and 55 refer to under what conditions we can ensure that the sender crashed and the communication channel is empty.

Unexpected behavior 1. From the logical viewpoint, we can rewrite Formula 51 into two separated formulas as the following:

$$\begin{aligned} \mathsf{constant} \geq 6 \times \mathit{\Phi} + \mathit{\Delta} \Rightarrow \mathsf{timeout} = \mathsf{constant} \qquad (56) \\ \wedge\ \mathsf{constant} < 6 \times \mathit{\Phi} + \mathit{\Delta} \Rightarrow \mathsf{timeout} \leq 6 \times \mathit{\Phi} + \mathit{\Delta} \qquad (57) \end{aligned}$$

However, from the implementation viewpoint, there exists a subtle difference between Formula 51 and the conjunction of Formulas 56 and 57. When Ivy tries to prove the inductive invariant, Ivy proves Formulas 56 and 57 separately. While Ivy can prove Formulas 56 and 57 successfully, this separation causes more difficulties to Ivy in proving the inductive invariant. As a result, Ivy cannot prove the following obligation in Invariant 71 that we will discuss in detail later.

$$(\mathsf{hLFSC} \geq (3 \times \mathit{\Phi} + \mathit{\Delta}) + \mathit{\Phi} \times \mathsf{timeout} \wedge \neg \mathsf{suspected}) \Rightarrow \mathsf{waitingTime} \geq \mathsf{timeout}$$

Hence, we need to use Formula 51, instead of Formulas 56 and 57. However, merging invariants is not always a good decision. More details are provided when we discuss Formulas 54 and 55 later.

Unexpected behavior 2. From the logical viewpoint, we can merge Formulas 54 and 55 into one formula as the following:

$$\begin{aligned} &(\mathsf{timeout} \geq 6 \times \mathit{\Phi} + \mathit{\Delta} \wedge \mathsf{waitingTime} \geq \mathsf{timeout}) \\ &\qquad \Rightarrow (\mathsf{sPC} = \mathsf{sCrash} \wedge \forall x \in \mathbb{Z} \,.\, \neg \mathsf{existsMsgOfAge}(x)) \qquad (58) \end{aligned}$$

While Ivy can prove the merged Formula 58, the merging causes more difficulties to Ivy. As a result, Ivy cannot prove the following obligation in Invariant 71.

$$(\mathsf{hLFSC} \geq (3 \times \mathit{\Phi} + \mathit{\Delta}) + \mathit{\Phi} \times \mathsf{timeout} \wedge \neg \mathsf{suspected}) \Rightarrow \mathsf{waitingTime} \geq \mathsf{timeout}$$

Hence, we need to write Formulas 54 and 55 separately. However, merging invariants is not always bad. We found that merging invariants can help Ivy in some cases. Section H lists unexpected behaviors of Ivy and Z3 that we faced in our experiments.

E.3 waitingTime, existsMsgOfAge and sTimer

This invariant is very similar to one in Section C.3, except that we need to add new constraints for cases where the sender crashes.

$$\begin{aligned}&\big((\forall x \in \mathbb{Z}.(0 \leq X \leq \varDelta) \Rightarrow \neg\mathsf{existsMsgOfAge}(x))\\ &\quad \wedge 0 < \mathsf{waitingTime} < 3 \times \varPhi\big)\\ \Rightarrow\ &\big((\mathsf{sPC} = \mathsf{sSnd} \wedge \mathsf{sTimer} \geq \mathsf{waitingTime}) \vee (\mathsf{sPC} = \mathsf{sCrash} \wedge \mathsf{sTimer} = 0)\big) \quad (59)\end{aligned}$$

E.4 waitingTime and the age of in-transit messages

This invariant is very similar to one in Section C.4, except that we need to add new constraints for cases where the sender crashes.

$$\begin{aligned}&(3 \times \varPhi \leq \mathsf{waitingTime} < 3 \times \varPhi + \varDelta) \Rightarrow\\ &\big((\mathsf{sPC} = \mathsf{sCrash} \wedge \forall x \in \mathbb{Z}\,.\,(0 \leq x \leq \varDelta) \Rightarrow \neg\mathsf{existsMsgOfAge}(x)) \vee\\ &\ (\exists x \in \mathbb{Z}\,.\,0 \leq x \leq \varDelta \wedge \mathsf{waitingTime} \leq 3 \times \varPhi + x \wedge \mathsf{existsMsgOfAge}(x))\big) \quad (60)\end{aligned}$$

E.5 waitingTime and the existence of old messages

This invariant is very similar to one in Section C.5, except that we need to add new constraints for cases where the sender crashes.

$$\begin{aligned}&\mathsf{waitingTime} \geq 3 \times \varPhi + \varDelta\\ \Rightarrow&\Big(\mathsf{existsMsgOfAge}(\varDelta) \vee\\ &\ \big(\mathsf{sPC} = \mathsf{sCrash} \wedge (\forall x \in \mathbb{Z}\,.\,0 \leq x \leq \varDelta \Rightarrow \neg\mathsf{existsMsgOfAge}(x))\big)\Big) \quad (61)\end{aligned}$$

E.6 waitingTime is unbound after the sender crashes

This invariant is in form of $A \vee B$ where:

- Subformula A refers to the cases where the sender crashes.

$$\begin{aligned}\Big(&(\forall x \in \mathbb{Z}\,.\,(0 \leq x \leq \varDelta) \Rightarrow \neg\mathsf{existsMsgOfAge}(x))\\ &\wedge\ \mathsf{waitingTime} \geq 3 \times \varPhi + \varDelta\\ &\wedge\ \mathsf{sPC} = \mathsf{sCrash}\Big) \quad (62)\end{aligned}$$

  Subformula A implies that waitingTime is unbound if the sender crashes.
- Subformula B is a conjunction of Invariants 30–32 in Section C.6, i.e. $((30) \wedge (31) \wedge (32))$. Subformula A ensures that every in-transit message is delivered before waitingTime reaches the guard $6 \times \varPhi + \varDelta$.

Unexpected bahavior 3. By type invariants, we can shorten the first condition in the subformula B as the following formula:

$$\big(\forall x \in \mathbb{Z} \,.\, \neg \mathsf{existsMsgOfAge}(x)\big) \tag{63}$$

However, if the replacement with Condition 63 is applied, Ivy cannot prove Invariant 71.

E.7 hLFSC and the age of in-transit messages

Invariant 64 describes the relationship between ages of in-transit messages and hLFSC when $\mathsf{hLFSC} < \Delta$. If the sender has crashed from hLFSC time-units, then no in-transit messages are younger than hLFSC time-units old.

$$\begin{aligned} & 0 \leq \mathsf{hLFSC} < \Delta \\ & \quad \Rightarrow \Big((\exists x \in \mathbb{Z} \,.\, 0 \leq x \leq \Delta \wedge \mathsf{existsMsgOfAge}(x)) \\ & \qquad \Rightarrow \big(\forall x \in \mathbb{Z} \,.\, (0 \leq x \leq \mathsf{hLFSC}) \Rightarrow \neg \mathsf{existsMsgOfAge}(x)\big)\Big) \end{aligned} \tag{64}$$

Explanation. Let t_0 be a time when the sender crashed and t be the current time. We have that $\mathsf{eval}(\mathsf{hLFSC}, t_0) = 0$ and that no new message has been sent from a time t_0. Hence, at a time t, there is no message that is younger than or equals hLFSC time-units old.
Invariant 65 describes the relationship between ages of in-transit messages and hLFSC when $\mathsf{hLFSC} \geq \Delta$. Invariant 65 refers to that if $\mathsf{hLFSC} \geq \Delta$, then either all in-transit messages are old or there exist no in-transit messages.

$$\begin{aligned} & \mathsf{hLFSC} \geq \Delta \Rightarrow \\ & \quad \Big(\big(\mathsf{existsMsgOfAge}(\Delta) \wedge \forall x \in \mathbb{Z} \,.\, (x \neq \Delta) \Rightarrow \neg \mathsf{existsMsgOfAge}(x)\big) \\ & \qquad \vee (\forall x \in \mathbb{Z}.(0 \leq x \leq \Delta) \Rightarrow \neg \mathsf{existsMsgOfAge}(x))\Big) \end{aligned} \tag{65}$$

E.8 All messages are delivered by $\mathsf{hLFSC} = 3 \times \Phi + \Delta$

Invariant 66 refers to that when $\mathsf{hLFSC} = 3 \times \Phi + \Delta$, all messages are delivered. In other words, the communication channel is empty.

$$\begin{aligned} & \mathsf{hLFSC} \geq 3 \times \Phi + \Delta \\ & \quad \Rightarrow (\forall x \in \mathbb{Z} \,.\, (0 \leq x \leq \Delta) \Rightarrow \neg \mathsf{existsMsgOfAge}(x)) \end{aligned} \tag{66}$$

In order to prove Invariant 66, we need three following Invariants 67–69 that play similar roles to Invariants 30–32 in Section C.6. The main idea is that the receiver must execute an instruction Receive at least once by a time when hLFSC reaches the guard $3 \times \Phi + \Delta$.

$$\begin{aligned}
&\big(\mathsf{hLFSC} \geq 2 \times \Phi + \Delta \\
&\quad \land \exists x \in \mathbb{Z} \,.\, 0 \leq x \leq \Delta \land \mathsf{existsMsgOfAge}(x)\big) \\
&\qquad \Rightarrow \big(\mathsf{rPC} = \mathsf{rNoSnd} \land \mathsf{rTimer} \geq \mathsf{hLFSC} - 2 \times \Phi - \Delta\big) && (67)\\
&\big(\Phi + \Delta \leq \mathsf{hLFSC} < 2 \times \Phi + \Delta \\
&\quad \land \exists x \in \mathbb{Z} \,.\, 0 \leq X \leq \Delta \land \mathsf{existsMsgOfAge}(x)\big) \\
&\qquad \Rightarrow \big((\mathsf{rPC} = \mathsf{rNoSnd} \lor \mathsf{rPC} = \mathsf{rComp}) \\
&\qquad\qquad \land \mathsf{rPC} = \mathsf{rComp} \Rightarrow \mathsf{rTimer} \geq \mathsf{hLFSC} - \Phi - \Delta\big) && (68)\\
&\big(\Delta \leq \mathsf{hLFSC} < \Phi + \Delta \\
&\quad \land \exists x \in \mathbb{Z} \,.\, 0 \leq X \leq \Delta \land \mathsf{existsMsgOfAge}(x)\big) \\
&\qquad \Rightarrow (\mathsf{rPC} = \mathsf{rRcv} \Rightarrow \mathsf{rTimer} \geq \mathsf{hLFSC} - \Delta) && (69)
\end{aligned}$$

- *Invariant 67.* Let t be the current time point when we have

$$\mathsf{eval}(\mathsf{hLFSC}, t) \geq 2 \times \Phi + \Delta$$

Let t_0 be the last time point when $\mathsf{eval}(\mathsf{waitingTime}, t_0) = \Delta$. By Invariant 65, we have $\mathsf{eval}(\mathsf{existsMsgOfAge}(\Delta), t_0) = \mathtt{true}$. We have $t - t_0 \geq 2 * \Phi$. Hence, in the time interval $[t_0, t]$ the receiver has to be active at least twice. Due to $\mathsf{eval}(\mathsf{waitingTime}, t) \geq 2 \times \Phi + \Delta$, the receiver has not executed any instructions Receive in the time interval $[t_0, t]$. It implies that the receiver executed an instruction Compute and then an instruction No Send. Hence, $\mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rNoSnd}$. Moreover, we know that the last time when the receiver is active is by $2 \times \Phi + \Delta$. Since both rTimer and hLFSC are measured with the global clock, we have that

$$\mathsf{eval}(\mathsf{rTimer}, t) \geq \mathsf{eval}(\mathsf{hLFSC}, t) - 2 \times \Phi - \Delta$$

- *Invariant 68.* Let t be the current time point when we have

$$\Phi + \Delta \leq \mathsf{eval}(\mathsf{waitingTime}, t) < 2 \times \Phi + \Delta$$

Let t_0 be the last time point when $\mathsf{eval}(\mathsf{waitingTime}, t_0) = \Delta$. By similar arguments in Invariant 67, we know that the receiver did not execute any instruction Receive in the time interval $[t_0, t]$. Moreover, in the time interval $[t_0, t]$, the receiver has to be active at least once. Hence, we have $\mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rNoSnd} \lor \mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rComp}$ at the time point t. Now assume $\mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rComp}$. Let t_R be the last time point when the receiver executed Receive. We have that t_R is before t_0, i.e., $t_R < t_0$ since $\mathsf{eval}(\mathsf{waitingTime}, t_0) = \Delta$ and $\mathsf{eval}(\mathsf{waitingTime}, t) \geq \Phi + \Delta$. By applying similar arguments in Invariant 26, we have that

$$\mathsf{eval}(\mathsf{rTimer}, t) \geq \mathsf{eval}(\mathsf{waitingTime}, t) - \Phi - \Delta$$

– *Invariant 69.* Let t be the current time point when we have

$$\Delta \leq \mathsf{eval}(\mathsf{waitingTime}, t) < \Phi + \Delta$$

Let t_0 be the last time point where $\mathsf{eval}(\mathsf{waitingTime}, t_0) = \Delta$. By similar arguments in Invariant 67, we know that the receiver did not execute any instruction Receive in the time interval $[t_0, t]$. Let t_R be the last time point when the receiver executed an instruction Receive. By similar arguments in Invariant 68, we know that t_R is before t_0, i.e., $t_R < t_0$. Now assume that $\mathsf{eval}(\mathsf{rPC}, t) = \mathsf{rRcv}$. By applying similar arguments in Invariant 26, we have that $\mathsf{eval}(\mathsf{rTimer}, t) \geq \mathsf{eval}(\mathsf{waitingTime}, t) - 3 \times \Phi - \Delta$.

E.9 waitingTime eventually reaches timeout after the sender crashes

Invariants 70–72 supports that waitingTime eventually reaches timeout after the sender crashes.

$$\begin{aligned}\Big(\big(&3 \times \Phi + \Delta < \mathsf{hLFSC} < (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout} \\ &\wedge \neg\mathsf{suspected} \\ &\wedge \mathsf{waitingTime} < \mathsf{timeout}\big) \Rightarrow \\ &\quad (\mathsf{hLFSC} \leq (3 \times \Phi + \Delta) + \Phi \times \mathsf{waitingTime} + \mathsf{rTimer})\Big)\end{aligned} \tag{70}$$

Explanation. Let t be the current time point when all premises in Invariant 70 hold, and t_0 be the time point when $\mathsf{eval}(\mathsf{hLFSC}, t_0) = 3 \times \Phi + \Delta$. We have $t - t_0 + \mathsf{eval}(\mathsf{hLFSC}, t_0) = \mathsf{eval}(\mathsf{hLFSC}, t)$ since hLFSC is measured with the global clock. By Invariant 66, the communication channel is empty at the time point t_0. Hence, the receiver will not receive any message from time t_0. It implies that variable waitingTime is increased by 1 whenever the receiver is active from time t_0. Moreover, we have that $\mathsf{eval}(\mathsf{waitingTime}, t_0) \geq 0$. Let k and ℓ be numbers such that $t = t_0 + k \times \Phi + \ell$ and $\ell < \Phi$. Since the receiver has to take at least one step in every time interval with Φ time-units, we have that $\mathsf{eval}(\mathsf{waitingTime}, t) \geq \mathsf{eval}(\mathsf{waitingTime}, t_0) + k$. If $\mathsf{eval}(\mathsf{waitingTime}, t) \geq \mathsf{eval}(\mathsf{waitingTime}, t_0) + k$ and $\mathsf{eval}(\mathsf{waitingTime}, t_0) = 0$, then we have $\ell = \mathsf{rTimer}$. Hence, the conclusion in Invariant 70 holds.
Invariant 71 is not to reason about reachable states. This invariant helps Ivy go through the inductive invariant by eliminating some counterexamples that are not reachable.

$$\Big((\mathsf{hLFSC} = (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout} - 1 \wedge \neg\mathsf{suspected}) \Rightarrow$$
$$\big((\mathsf{waitingTime} = 6 \times \Phi + \Delta - 1 \wedge \mathsf{rTimer} = \Phi - 1)$$
$$\vee (\mathsf{waitingTime} = 6 \times \Phi + \Delta))\big)\Big) \wedge$$
$$\Big((\mathsf{hLFSC} \geq (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout} \wedge \neg\mathsf{suspected}) \Rightarrow$$
$$\mathsf{waitingTime} \geq \mathsf{timeout}\Big) \tag{71}$$

Invariant 72 is based on Invariant 71 and says that if hLFSC is greater than the guard $(3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout}$ and if the receiver still trusts the sender, then we have $\mathsf{waitingTime} \geq \mathsf{timeout}$.

$$\Big((\mathsf{hLFSC} > (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout} \wedge \neg\mathsf{suspected}) \Rightarrow$$
$$(\mathsf{waitingTime} \geq \mathsf{timeout})\Big) \tag{72}$$

E.10 The receiver eventually suspects the crashed sender

Let t_0 be the time point when $\mathsf{hLFSC} = 3 \times \Phi + \Delta$. By Invariant 66, there are no in-transit messages from t_0. Hence, $\mathsf{waitingTime}$ will not be reset to 0 from t_0. We also have $\mathsf{eval}(\mathsf{waitingTime}, t_0) \geq 0$. Let t_1 be the current time point when $\mathsf{hLFSC} = 3 \times \Phi + \Delta + \Phi \times \mathsf{timeout}$. We have $\mathsf{eval}(\mathsf{waitingTime}, t_1) \geq \mathsf{timeout}$ by Invariants 71 and 72. However, the receiver might still trust the sender at t_1. Three following Invariants 73–75 ensure that the when hLFSC reaches the guard $3 \times \Phi + \Delta + \Phi \times \mathsf{timeout} + 3 \times \Phi$ at some time $t \geq t_1$, then the receiver already suspects the crashed sender by time t. Invariants 73–75 are based on similar arguments of Invariants 30–32, except that we focus on instructions Compute. We denote $G = 3 \times \Phi + \Delta + \Phi \times \mathsf{timeout}$.

$$\Big(((\mathsf{hLFSC} \geq G + 2 \times \Phi) \wedge \neg\mathsf{suspected}) \Rightarrow$$
$$(\mathsf{rPC} = \mathsf{rRcv} \wedge$$
$$\mathsf{rTimer} \geq \mathsf{hLFSC} - (3 \times \Phi + \Delta + \Phi \times \mathsf{timeout}) - 2 \times \Phi)\Big) \tag{73}$$
$$\Big(((G + \Phi \leq \mathsf{hLFSC} < G + 2 \times \Phi) \wedge \neg\mathsf{suspected}) \Rightarrow$$
$$((\mathsf{rPC} = \mathsf{rNoSnd} \vee \mathsf{rPC} = \mathsf{rRcv}) \wedge$$
$$(\mathsf{rPC} = \mathsf{rNoSnd} \Rightarrow \mathsf{rTimer} \geq \mathsf{hLFSC} - G - \Phi))\Big) \tag{74}$$
$$\Big(((G \leq \mathsf{hLFSC} < G + \Phi) \wedge \neg\mathsf{suspected}) \Rightarrow$$
$$((\mathsf{rPC} = \mathsf{rNoSnd} \vee \mathsf{rPC} = \mathsf{rComp} \vee \mathsf{rPC} = \mathsf{rRcv}) \wedge$$
$$(\mathsf{rPC} = \mathsf{rComp} \Rightarrow \mathsf{rTimer} \geq \mathsf{hLFSC} - G))\Big) \tag{75}$$

F An inductive invariant with Strong Completeness and big $\mathsf{timeout}$

In this section, we present an inductive invariant for the liveness property Strong Completeness in cases that $\mathsf{timeout}$ is initialized with a value that is at least $6 \times \Phi + \Delta$, i.e., $\mathsf{eval}(\mathit{timeout}, 0) \geq 6 \times \Phi + \Delta$. The inductive invariant in Section F looks much simpler than the inductive invariant in Section E. The reason is that we need only one ghost variable $\mathsf{constant}$ to keep track of whether $\mathsf{timeout}$ is unchanged.
An important point is that we use Invariant 97, instead of Invariant 71. Invariant 97 focuses on reachable states, and is more readable. However, this replacement requires strengthening steps to help Ivy go through the inductive invariant. Initially, we have that $\mathsf{constant} = \mathsf{timeout}$ and $\mathsf{hLFSC} = -1$ and the sender is correct. Notice that by the inductive invariant with the safety property Strong Accuracy in Section C, the receiver never has a wrong suspicion of the correctness of the sender.

F.1 Type invariants

The following contains type invariants.

$$0 \leq \Delta \wedge 1 \leq \Phi \tag{76}$$
$$\mathsf{sPC} = \mathsf{sSnd} \vee \mathsf{sPC} = \mathsf{sCrash} \tag{77}$$
$$\mathsf{rPC} = \mathsf{rNoSnd} \vee \mathsf{rPC} = \mathsf{rRcv} \vee \mathsf{rPC} = \mathsf{rComp} \tag{78}$$
$$0 \leq \mathsf{sTimer} \wedge \mathsf{sTimer} < 3 \times \Phi \tag{79}$$
$$0 \leq \mathsf{rTimer} \wedge \mathsf{rTimer} < \Phi \tag{80}$$
$$\mathsf{timeout} \geq 6 \times \Phi + \Delta \tag{81}$$
$$\mathsf{timeout} = \mathsf{constant} \tag{82}$$
$$0 \leq \mathsf{waitingTime} \tag{83}$$
$$\forall x \in \mathbb{Z} . (x < 0 \vee \Delta < x) \Rightarrow \neg \mathsf{existsMsgOfAge}(x) \tag{84}$$
$$\forall x \in \mathbb{Z} . \mathsf{tmpExistsMsgOfAge}(x) = \mathsf{existsMsgOfAge}(x) \tag{85}$$
$$-1 \leq \mathsf{hLFSC} \tag{86}$$
$$\mathsf{sPC} = \mathsf{sCrash} \Leftrightarrow (\mathsf{sTimer} = 0 \wedge \mathsf{hLFSC} \geq 0) \tag{87}$$
$$\mathsf{sPC} = \mathsf{sSnd} \Leftrightarrow (\mathsf{waitingTime} < \mathsf{timeout} \wedge \mathsf{hLFSC} = -1 \wedge \neg \mathsf{suspected}) \tag{88}$$

F.2 Strong completeness with initialized big $\mathsf{timeout}$

Under our assumptions and encoding, the liveness property Strong Completeness can be rewritten as the following:

$$\mathsf{hLFSC} \leq (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout} + (3 \times \Phi) \tag{89}$$
$$\wedge\ \mathsf{hLFSC} = (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout} + (3 \times \Phi) \Rightarrow \mathsf{suspected} \tag{90}$$

$$\wedge\ \mathsf{hLFSC} = (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout} + (3 \times \Phi) \Rightarrow \mathsf{sPC} = \mathsf{sCrash} \tag{91}$$

$$\wedge\ \mathsf{suspected} \Rightarrow \big(\mathsf{sPC} = \mathsf{sCrash} \wedge \mathsf{waitingTime} \geq \mathsf{timeout} \wedge \forall x \in \mathbb{Z} \,.\, \neg \mathsf{existsMsgOfAge}(x)\big) \tag{92}$$

Merging Invariants 90 and 91 causes more difficulties, and Ivy cannot prove the inductive invariant.

F.3 waitingTime, existsMsgOfAge and sTimer

Invariant 59 is used.

F.4 waitingTime and ages of in-transit messages

Invariant 60 is used.

F.5 waitingTime and existing old messages

Invariant 61 is used. In addition, we need to use additional Invariant 93. While Invariant 61 implies Invariant 93, listing Invariant 93 explicitly helps Ivy prove the inductive invariant.

$$\mathsf{waitingTime} = 3 \times \Phi \Rightarrow \big((\mathsf{sPC} = \mathsf{sCrash} \wedge (\forall x \in \mathbb{Z} \,.\, (0 \leq x \leq \Delta) \Rightarrow \neg \mathsf{existsMsgOfAge}(x))) \vee (\exists x \in \mathbb{Z} \,.\, 0 \leq x \leq \Delta \wedge \mathsf{existsMsgOfAge}(x))\big) \tag{93}$$

F.6 waitingTime is unbound after the sender crashes

Invariant in Section E.6 is used.

F.7 hLFSC and ages of in-transit messages

Instead of Invariant 64, we use the following Invariant 94. Invariant 65 is used.

$$0 \leq \mathsf{hLFSC} < \Delta \Rightarrow \big(\forall x \in \mathbb{Z} \,.\, (0 \leq x \leq \mathsf{hLFSC}) \Rightarrow \neg \mathsf{existsMsgOfAge}(x)\big) \tag{94}$$

F.8 All messages are delivered by $\mathsf{hLFSC} = 3 \times \Phi + \Delta$

Invariants 66–69 are used.

F.9 waitingTime eventually reaches timeout after the sender crashes

Invariants 95–96 are similar to those in Section E.9. Since timeout is initially greater than or equals $6 \times \Phi + \Delta$, the receiver never has a wrong suspicion of the correctness of the sender. Hence, the premises of Invariants 95 and 96 are simpler than those of Invariants 70 and 72, respectively. In Invariant 97, we need to mention $timeout \geq 6 \times \Phi + \Delta$ to help Ivy.

$$\begin{aligned}
&\Big((\mathsf{hLFSC} > 3 \times \Phi + \Delta \\
&\quad \wedge \mathsf{hLFSC} < (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout}\big) \Rightarrow \\
&\qquad (\mathsf{hLFSC} \leq (3 \times \Phi + \Delta) + \Phi \times \mathsf{waitingTime} + \mathsf{rTimer})\Big) \qquad (95)
\end{aligned}$$

$$\begin{aligned}
&\Big((\mathsf{hLFSC} > (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout}) \Rightarrow \\
&\qquad (\mathsf{waitingTime} \geq \mathsf{timeout})\Big) \qquad (96)
\end{aligned}$$

Invariant 97 works like a bridge between Invariants 96 and 96.

$$\begin{aligned}
&\mathsf{timeout} \geq 6 \times \Phi + \Delta \\
&\wedge \Big((\mathsf{hLFSC} = (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout} - 1 \\
&\qquad \wedge \mathsf{waitingTime} \leq \mathsf{timeout}) \Rightarrow \\
&\qquad ((\mathsf{waitingTime} = \mathsf{timeout} - 1 \wedge \mathsf{rTimer} = \Phi - 1) \\
&\qquad\quad \vee (\mathsf{waitingTime} = \mathsf{timeout}))\Big) \wedge \\
&\wedge \Big((\mathsf{hLFSC} \geq (3 \times \Phi + \Delta) + \Phi \times \mathsf{timeout}) \Rightarrow \\
&\qquad\quad \mathsf{waitingTime} \geq \mathsf{timeout}\Big) \qquad (97)
\end{aligned}$$

F.10 The receiver eventually suspects the crashed sender

Invariants 73–75 are used.

G An inductive invariant with Strong Completeness and fixed timeout

All invariants in Section F are used, except that a type invariant $\mathsf{timeout} = 6 \times \Phi + \Delta$ replaces a type invariant $\mathsf{timeout} \geq 6 \times \Phi + \Delta$.

H Unexpected behaviors of Ivy and Z3

In this section, we describe unexpected behaviors of Ivy, and its backend solver Z3 that we faced in our experiments. Importantly, our Ivy specifications are not in decidable theories that Ivy supports. When reasoning about decidable specifications, Ivy is more stable, predictable and transparent.

H.1 Update a relation

We first tried to encode the application of the shift-left operator on the relation existsMsgOfAge with the following Ivy code.

$$\mathsf{existsMsgOfAge}(X + 1) := \mathsf{existsMsgOfAge}(X) \tag{98}$$

We expected that the transformation of Line 98 to SMT would have the following semantics:

$$\forall X : \mathsf{num}\ .\ \mathsf{existsMsgOfAge}'(X + 1) = \mathsf{existsMsgOfAge}(X)$$

where existsMsgOfAge′ refers the value of the relation existsMsgOfAge in the next state. However, the transformation of Line 98 to SMT produces the following constraint:

$$\begin{aligned}\forall X, Y : \mathsf{num}.(&Y = X + 1 \Rightarrow \mathsf{existsMsgOfAge}'(X + 1) = \mathsf{existsMsgOfAge}(X))\\ \vee(&Y \neq X + 1 \Rightarrow \mathsf{existsMsgOfAge}'(X + 1) = \mathsf{existsMsgOfAge}(X))\end{aligned}$$

H.2 Merge invariants

Ivy proves invariants separately and step-by-step. Hence, merging invariants makes Ivy prove bigger obligations. Bigger obligations usually cause more difficulties to reason about an inductive invariant, e.g., Invariants 54 and 55 in Section E.
Surprisingly, we observed that a bigger obligation can help Ivy go through an inductive invariant in some cases. For example, see Invariant 51 in Section E. We conjecture that Invariant 51 reminds Ivy to focus on constraints with the guard $6 \times \Phi + \Delta$.

H.3 The order of invariants

While changing the order of invariants produces the same inductive invariant, we observed that a good order of invariants can help Ivy go through an inductive invariant. For example, we present Invariant 72 before Invariant 71 in this report and in the corresponding Ivy specification named "fd_sc.ivy". With this order, Ivy can go through the inductive invariant. We tried to move Invariant 72 up, but then Ivy could not prove the inductive invariant.
We also observed that some order of invariants can help Ivy prove an inductive invariant faster. For example, Invariants 66–69.
We conjecture that if proving an invariant Inv_1 requires an invariant Inv_2, then Inv_2 should appear before Inv_1. We follow this idea when writing our specification.
Finally, invariants in our specifications usually appear in the order that we present in this report, except the specification "fd_sc_big_timeout.ivy". In the specification "fd_sc_big_timeout.ivy", Invariants presenting the propery Strong Completeness are provided before Invariants ensuring that $\mathsf{waitingTime}$ eventually reaches $\mathsf{timeout}$ after the sender crashes. (We tried to put those invariants in other places but Ivy cannot go through an inductive invariant.)

H.4 Invariants as intermediate proof-steps

We now discuss Invariant 97. Its first constraint $\mathsf{timeout} \geq 6 \times \Phi + \Delta$ already appears in type invariants. The second conjunction is a specific case in Invariant 95. From the theoretical viewpoint, these constraints are unnecessary. However, if we remove them, Ivy cannot go through an inductive invariant.
Another example is Invariant 93. While Invariant 61 implies Invariant 93, listing Invariant 93 explicitly helps Ivy prove the inductive invariant.