Analyzing Paxos with Fault-Tolerant Multiparty Session Types

Bachelor thesis by Nicolas Daniel Torres
Date of submission: November 23, 2021

1. Review: Prof. Dr. Kirstin Peters
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

Computer Science Department
<institute>
<working group>

Contents

1 Introduction

In distributed systems components on different computers coordinate and communicate via message passing to achieve a common goal. Sometimes, to achieve this goal, the individual components need to reach consensus, i.e., agree on the value of some data using a consensus algorithm. For example in state machine replication or when deciding which database transactions should be committed in what order. For such a distributed system to behave correctly the consensus algorithm needs to be correct. Thus, analyzing consensus algorithms is important.

To achieve consensus, consensus algorithms must satisfy the following properties: termination, validity, and agreement [2]. Proving these properties can be complicated. Model checking tools lead to big state-spaces so static analysis is preferable. For static analysis Multiparty Session Types are particularly interesting because session typing can ensure protocol conformance and the absence of communication errors and deadlocks [6].

Due to the presence of faulty processes and unreliable communication consensus algorithms are designed to be fault-tolerant. Modelling fault-tolerance is not possible using Multiparty Session Types, thus a fault-tolerant extension is necessary. Peters, Nestmann, and Wagner developed such an extension called Fault-Tolerant Multiparty Session Types.

In this work we will use Fault-Tolerant Multiparty Session Types to analyze the consensus algorithm Paxos, as described in [4].

2 Technical Preliminaries

First, we define the sorts, some additional notation, and use them to define the global type. Afterwards we define some sets and functions to create the processes.

3 Model

First, we specify some sorts with which we can then define the global type. Afterwards, we define the processes for the proposer and the acceptor. Finally, we will study an example run of the model.

3.1 Sorts

Sorts are basic data types. We assume the following sorts.

$\text{Maybe } a = \text{Just } a \mid \text{Nothing}$

$\text{Bool} = \{\text{true}, \text{false}\}$

We assume a set of values Value.

$\text{Proposal } a = \text{Proposal } \mathbb{N}\, a$

$\text{Promise } a = \text{Promise Maybe Proposal } a \mid \text{Nack } \mathbb{N}$

3.2 Global Type

Since each proposer initiates its own session the global type can be defined for one proposer. A quorum of acceptors A_Q is assumed.

The last phase of Paxos contains no inter-process communication, so it is not modeled in the global type.

$$G_{p,A_Q} = (\mu X)\left(\bigodot_{a \in A_Q}\ p \to_u a : l1a\,\langle \mathbb{N} \rangle\right).\left(\bigodot_{a \in A_Q}\ a \to_u p : l1b\,\langle \text{Promise Value} \rangle\right).$$
$$p \to_w A_Q :$$
$$\quad \mathit{Accept}.\left(\bigodot_{a \in A_Q}\ p \to_u a : l2a\,\langle \text{Proposal Value} \rangle\right).\mathit{end} \oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end}$$

We can distinguish the individual phases of the Paxos algorithm by the labels $l1a$, $l1b$, and $l2a$.

In the first two steps, $1a$ and $1b$, the proposer sends its proposal number to each acceptor in A_Q and listens for their responses. In step 2a the proposer decides whether to send an Accept or $\mathit{Restart}$ message to restart the algorithm. This decision is broadcast to all acceptors in A_Q. Should the proposer crash the algorithm ends for this particular proposer and quorum of acceptors.

3.3 Functions

We define some functions which we use in the next section to define the processes.

$\text{proposalNumber} : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$\text{proposalNumber}\,(a, b)$ returns a proposal number when given two natural numbers.

$\text{promiseValue} : \texttt{list of}\ \text{Promise}\ a \to a$

$\text{promiseValue}\,(ps)$ returns a fresh value if none of the promises in ps contain a value. Otherwise, the best value is returned. Usually, that means the value with the highest associated proposal number. A promise contains a value v if it is of the form $\text{Promise}\ (\text{Just}\ v)$.

$\text{anyNack} : \texttt{list of}\ \text{Promise}\ a \to \text{Bool}$
$\text{anyNack}\,([]) = \text{false}$
$\text{anyNack}\,((\text{Nack}\ \_\#\_)) = \text{true}$
$\text{anyNack}\,((\_\#xs)) = \text{anyNack}\,(xs)$

$\text{anyNack}\,(ps)$ returns true if the list contains at least one promise of the form $\text{Nack}\ n$. Otherwise, it returns false.

$\text{promiseCount} : \texttt{list of}\ \text{Promise}\ a \to \mathbb{N}$
$\text{promiseCount}\,([]) = 0$
$\text{promiseCount}\,((\text{Promise}\ \_\#xs)) = 1 + \text{promiseCount}\,(xs)$
$\text{promiseCount}\,((\_\#xs)) = \text{promiseCount}\,(xs)$

$\mathrm{promiseCount}\,(ps)$ takes a list of promises ps and calculates the number of promises in that list of that have the form $\mathrm{Promise}\ m$.

$\mathrm{gt} : a \to \mathrm{Maybe}\ a \to \mathrm{Bool}$
$\mathrm{gt}\,(\_, \mathrm{Nothing}) = \mathrm{true}$
$\mathrm{gt}\,(a, \mathrm{Just}\ b) = a > b$

$\mathrm{ge} : a \to \mathrm{Maybe}\ a \to \mathrm{Bool}$
$\mathrm{ge}\,(\_, \mathrm{Nothing}) = \mathrm{true}$
$\mathrm{ge}\,(a, \mathrm{Just}\ b) = a \geq b$

$\mathrm{nFromProposal} : \mathrm{Proposal}\ a \to \mathbb{N}$
$\mathrm{nFromProposal}\,(\mathrm{Proposal}\ n\ \_) = n$

$\mathrm{nFromProposal}\,(p)$ retrieves the proposal number n inside proposal p, which has the form $\mathrm{Proposal}\ n\ pr$.

$\mathrm{genA_Q} : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \to \mathbb{N}$

$\mathrm{genA_Q}\,(i, ac, pc)$ returns a randomly selected set A_Q with $A_Q \subseteq A = \{1, \ldots, ac\}$ and $|A_Q| > \frac{|A|}{2}$. A_Q consists of any majority of acceptors. In Paxos a majority of acceptors forms a quorum, i.e. an accepting set with which a value can be chosen [3].

$\mathrm{update}\,(n, m)$ replaces the value inside n with the value in m.

3.4 Processes

3.4.1 System Initialization

$$\begin{aligned}&\mathrm{Sys}\,(ac, pc) = \overline{a}\,[2]\,(t)\,.\,\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\,(ac+1, \mathrm{genA_Q}\,(ac+pc, ac, pc)\,, ac+pc, ac+pc, [])\\ &\mid a\,[1]\,(t)\,.\Pi_{ac<i<ac+pc}\ \mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\,(ac+1, \mathrm{genA_Q}\,(i, ac, pc)\,, i, i, [])\\ &\mid \Pi_{1\leq j\leq ac}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(j, ac+1, ac, pc, n_j, pr_j)\end{aligned}$$

$$\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\left(i, A_Q, n, m, \overrightarrow{V}\right) = \overline{b_n}\,[i]\,(s)\,.\,\mathrm{P^p}$$

$$\mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(j, i, ac, pc, n, pr) = \Pi_{ac<k\leq ac+pc}\ b_k\,[j]\,(s)\,.\,\mathrm{P^a}$$

$\mathrm{Sys}\,(ac, pc)$, $\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\left(i, A_Q, n, m, \overrightarrow{V}\right)$, and $\mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(j, i, ac, pc, n, pr)$ describe the system initialization. ac and pc are the number of acceptors and proposers respectively.

An outer session is created through shared-point a. This outer session is not strictly necessary but was left in to allow for easier extension of the model. The acceptors are initialized using indices from 1 to ac and the proposers are initialized using indices from $ac+1$ to $ac+pc$.

$\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\left(i, A_Q, n, m, \overrightarrow{V}\right)$ is initialized with the proposer's role in its own session i, which is always $ac+1$, a quorum of acceptors A_Q, an index n, and a vector $\overrightarrow{V}$. Each proposer has the same role $i = ac+1$ but uses a different shared-point b_n according to its index n. $\overrightarrow{V}$ is used in the proposer to collect and evaluate the responses from the acceptors. It is always initialized with an empty list $[]$. Shared-point b_n is used to initiate a session. Afterwards, the process behaves like P^{p}.

$\mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(j, i, ac, pc, n, pr)$ is initialized with the acceptor's index j, the proposer index i, which is always $ac+1$, ac, pc, initial knowledge for the highest promised proposal number n, if available, and initial knowledge for the most recently accepted proposal pr, if available. n is of type $\mathrm{Maybe}\ \mathbb{N}$ and pr is of type $\mathrm{Maybe}\ (\mathrm{Proposal}\ \mathrm{Value})$ thus both can be $\mathrm{Nothing}$. Each of the proposers' session requests are accepted in a separate subprocess. These subprocesses run parallel to each other but still access the same values for n and pr. We observe that each subprocess in an acceptor accesses a different channel s, since it is generated by the proposer and passed through when the proposers' session request is accepted. Afterwards, each subprocess behaves like P^{a}.

3.4.2 Proposer

$$
\begin{aligned}
&\mathrm{P}^{\mathrm{p}} = (\mu X)\,\mathrm{update}\,(n, n+1)\,.\\
&\quad\left(\textstyle\bigodot_{j\in A_Q}\ s\,[i,j]!_u l1a\,\langle \mathrm{proposalNumber}\,(n,m)\rangle\right).\\
&\quad\left(\textstyle\bigodot_{j\in A_Q}\ s\,[j,i]?_u l1b\,\langle\bot\rangle\,(v_j)\right).\\
&\quad\text{if anyNack}\left(\overrightarrow{V}\right)\ \text{or promiseCount}\left(\overrightarrow{V}\right) < \left\lceil\tfrac{i}{2}\right\rceil\\
&\qquad\text{then } s\,[i, A_Q]!_w Restart.X\\
&\qquad\text{else}\\
&\qquad\ s\,[i, A_Q]!_w Accept.\\
&\qquad\ \textstyle\bigodot_{j\in A_Q}\ s\,[i,j]!_u l2a\left\langle \mathrm{Proposal}\ \ \mathrm{proposalNumber}\,(n,m)\ \ \mathrm{promiseValue}\left(\overrightarrow{V}\right)\right\rangle.\\
&\qquad end
\end{aligned}
$$

At the start of the recursion n is incremented to make sure every run of the recursion uses a different n and thus a different proposal number. The proposal number is sent to every

acceptor in A_Q and their replies are gathered in $\overrightarrow{V}$ through v_j. Because $i = ac + 1$, the minimum number of acceptors needed to form a majority is $\left\lceil \frac{i}{2} \right\rceil = \left\lceil \frac{ac+1}{2} \right\rceil$. If any $\mathrm{Nack}\ x$ was received or the number of $\mathrm{Promise}\ y$ received is less than that needed for the smallest majority the proposer restarts the algorithm. Otherwise, the proposer sends its proposal to the acceptors and terminates.

3.4.3 Acceptor

$$
\begin{array}{l}
\mathrm{P}^{\mathrm{a}} = (\mu X)\, s\,[i,j]?_u l1a\,\langle \bot \rangle\,(n')\,. \\
\quad \mathrm{if}\ \ n' = \bot \\
\quad\quad \mathrm{then}\ \ \mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} \\
\quad\quad \mathrm{else} \\
\quad\quad\quad \mathrm{if}\ \ \mathrm{gt}\,(n', n) \\
\quad\quad\quad \mathrm{then}\ \ \mathrm{update}\,(n, n')\,.s\,[j,i]!_u l1b\,\langle \mathrm{Promise}\ \ pr \rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} \\
\quad\quad\quad \mathrm{else}\ \ s\,[j,i]!_u l1b\,\langle \mathrm{Nack}\ \ n \rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}
\end{array}
$$

$$
\begin{array}{l}
\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} = s\,[i,j]?_w \mathit{Accept}.s\,[i,j]?_u l2a\,\langle \bot \rangle\,(pr')\,. \\
\quad \mathrm{if}\ \ pr' = \bot \\
\quad\quad \mathrm{then}\ \ \mathit{end} \\
\quad\quad \mathrm{else} \\
\quad\quad\quad \mathrm{if}\ \ \mathrm{ge}\,(\mathrm{nFromProposal}\,(pr'), n) \\
\quad\quad\quad\quad \mathrm{then}\ \ \mathrm{update}\,(pr, pr')\,.\,\mathrm{update}\,(n, \mathrm{Just}\ \ \mathrm{nFromProposal}\,(pr'))\,.\mathit{end} \\
\quad\quad\quad\quad \mathrm{else}\ \ \mathit{end} \\
\quad \oplus\, \mathit{Restart}.X \\
\quad \oplus\, \mathit{Abort}.\mathit{end}
\end{array}
$$

For each proposer an acceptor has a corresponding subprocess, which behaves like P^{a}. These subprocesses access the same values for n and pr. This means that updating these values with $\mathrm{update}\,(n, m)$ updates them for all subprocesses of an acceptor.

Each subprocess can communicate with one proposer. Thus, if that proposer does not or can not communicate with a particular subprocess of an acceptor then there is no need for that subprocess. It is possible that an acceptor participates in a proposers' session but is not contained in the proposers' quorum of acceptors A_Q, in which case the proposer does not communicate with that acceptor. It is also possible for a proposer to crash or otherwise terminate, in which case the proposer can not communicate with that acceptor.

Each subprocess starts out by potentially receiving a proposal number n' from the corresponding proposer. If the acceptor does receive a proposal number n' it responds with either $\mathrm{Promise}\ pr$ or $\mathrm{Nack}\ n$, depending on the values of n' and n. If the acceptor does not receive a proposal number then it sends no response to the proposer. In either case the subprocess moves on to receive the proposers' decision in phase $2a$.

Since the proposers' decision broadcast is weakly reliable, there are two cases in which the acceptor receives no decision. The proposer might have terminated or this particular acceptor is not in the proposers' quorum of acceptors A_Q. In either case this particular subprocess of the acceptor is no longer needed, because each subprocess of the acceptor exclusively communicates with one proposer. Thus, the subprocess terminates in the default branch $Abort$.

In the $Restart$ branch this particular subprocess of the acceptor restarts the algorithm to match the corresponding proposer.

In the $Accept$ branch the acceptor potentially receives a proposal pr' from the corresponding proposer. The acceptor updates n and pr if the proposal number in pr' is greater or equal to n. Then the subprocess terminates. If the acceptor does not receive a proposal or the proposal number of pr' is less than n the subprocess terminates without updating n or pr.

3.5 Failure Patterns

Chandra and Toueg introduce a class of failure detectors $\Diamond\mathscr{W}$, which is called *eventually weak* in [1]. Failure detectors in $\Diamond\mathscr{W}$ satisfy the following properties: (1) eventually every process that crashes is permanently suspected by some correct process and (2) eventually some correct process is never suspected by any correct process.

In all three phases modeled in the global type it is possible to suspect senders. In phases $1a$ and $2a$, with labels $l1a$ and $l2a$ respectively, the acceptors may suspect some proposers. The proposers may suspect some acceptors in phase $1b$ with label $l1b$. Accordingly, $\mathtt{FP}_{\mathtt{uskip}}$ is implemented with a failure detector in $\Diamond\mathscr{W}$ for phases $1a$, $1b$, and $2a$.

Similarly, message loss is possible in all phases modeled in the global type. Thus, $\mathtt{FP}_{\mathtt{ml}}$ is also implemented with a failure detector in $\Diamond\mathscr{W}$.

For the weakly reliable broadcast in phase $2a$, the failure pattern $\mathtt{FP}_{\mathtt{wskip}}$ returns true if, and only if, the corresponding proposer crashed or otherwise terminated.

For Paxos to work a majority of acceptors needs to be alive. That means that the number of failed acceptors f needs to satisfy $n > 2f$ where n is the total number of acceptors, except in one case where there are 2 acceptors. Then, at most one acceptor may crash [3]. $\texttt{FP}_{\texttt{crash}}$ returns true if, and only if, at least one more acceptor may crash, i.e. $n > 2(f + 1)$ is satisfied.

In Paxos there is no need to reject outdated messages so $\texttt{FP}_{\texttt{uget}}$ is implemented with a constant true.

3.6 Example

In this section we will study an example run of the model with 3 acceptors and 2 proposers. First, we will take a look at the example scenario. Then we will examine the scenario using reduction rules starting at system initialization.

3.6.1 Scenario

Figure 3.1 provides an overview where A_1, A_2, and A_3 are the acceptors and P_4 and P_5 are the proposers. In steps (1) to (5), P_5 completes the Paxos algorithm with A_2 and A_3 and terminates.

At this point A_2 has promised not to accept any proposal numbered less than 10 and has accepted the value abc. So, when P_4 tries to use 4 as its proposal number (6), it receives $\mathrm{Nack}\ 10$ from A_2 (8) and has to restart the algorithm (9).

P_4 then runs through the Paxos algorithm with A_1 and A_2 starting with a new prepare request (10) with a higher proposal number. In step (12) P_4 learns that value abc with proposal number 10 has already been accepted by A_2. Later, in step (14), P_4 issues a proposal with the value of the highest-numbered proposal that it receives as a response to its prepare request. In this case there is only one such proposal, which is $\mathrm{Proposal}\ 10\ \mathrm{abc}$.

In the end all 3 acceptors have accepted the value abc. A_1 and A_2 have accepted $\mathrm{Proposal}\ 15\ \mathrm{abc}$ and A_3 has accepted $\mathrm{Proposal}\ 10\ \mathrm{abc}$.

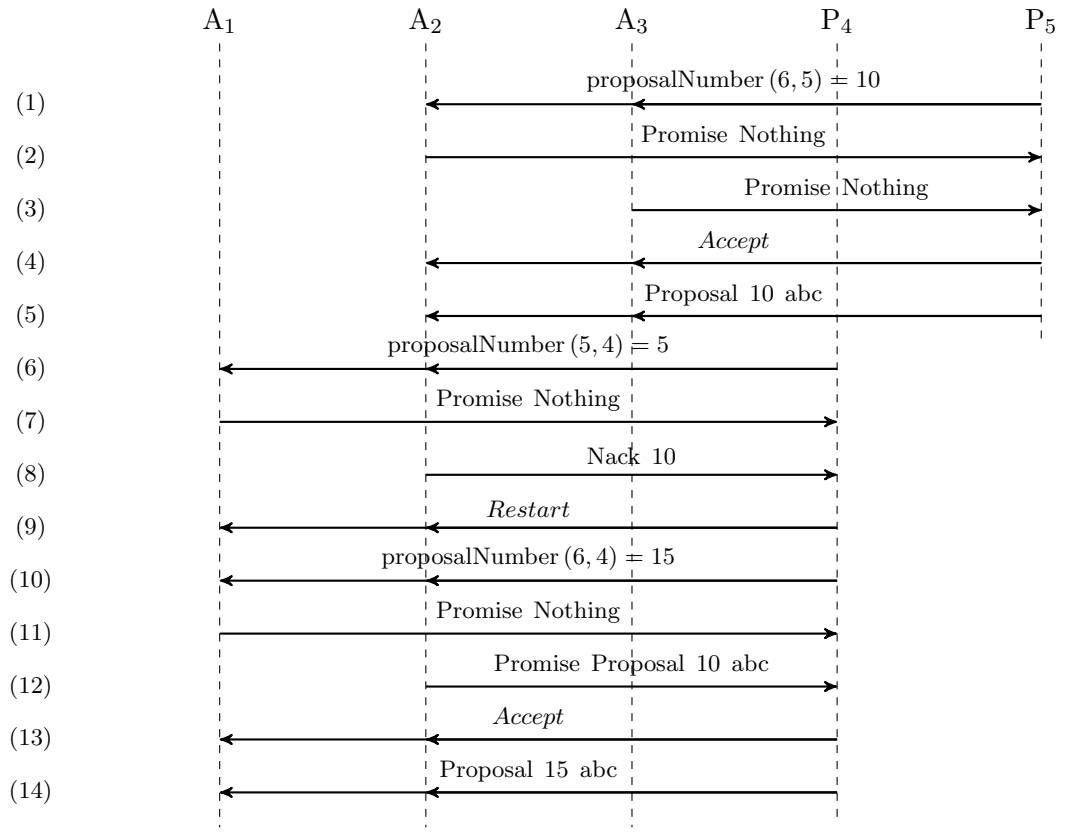

Figure 3.1: Example scenario with 3 acceptors and 2 proposers.

3.6.2 Formulae

We set $ac = 3$, $pc = 2$, and $V = \{\mathrm{abc}, \mathrm{def}, \ldots, \mathrm{vwx}, \mathrm{yz}\}$.

System Initialization

After inserting ac and pc and applying (Init) once we have:

$$\begin{aligned}
&\mathrm{Sys}\,(ac, pc) = \mathrm{Sys}\,(3,2) = a\,[1]\,(t)\,.\Pi_{3<i<5}\ \mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\,(4, \mathrm{genA_Q}\,(i,3,2)\,, i, i, [])\\
&\mid \overline{a}\,[2]\,(t)\,.\,\mathrm{P}^{\mathrm{p}}_{\mathrm{init}}\,(4, \mathrm{genA_Q}\,(5,3,2)\,, 5, 5, [])\\
&\mid \Pi_{1\le j\le 3}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{init}}\,(j, 4, 3, 2, n_j, pr_j)
\end{aligned}$$

$$\begin{aligned}
&\longmapsto^{*} (\nu t)\,\left(\overline{b_4}\,[4]\,(s)\,.\,\mathrm{P^p}\ = \mathrm{P}_4\right.\\
&\mid \overline{b_5}\,[4]\,(r)\,.\,\mathrm{P^p}\ = \mathrm{P}_5\\
&\mid\ (b_4\,[1]\,(s)\,.\,\mathrm{P^a}\ \mid b_5\,[1]\,(r)\,.\,\mathrm{P^a})\ = \mathrm{A}_1\\
&\mid\ (b_4\,[2]\,(s)\,.\,\mathrm{P^a}\ \mid b_5\,[2]\,(r)\,.\,\mathrm{P^a})\ = \mathrm{A}_2\\
&\mid\ (b_4\,[3]\,(s)\,.\,\mathrm{P^a}\ \mid b_5\,[3]\,(r)\,.\,\mathrm{P^a})\ = \mathrm{A}_3\\
&\left.\mid \Pi_{1\le k,l\le 2, k\ne l}\ t_{k\to l} : []\right)
\end{aligned}$$

We apply (Init) thrice for shared-point b_4 and thrice again for shared-point b_5 to obtain:

$$\begin{aligned}
&\longmapsto^{*} (\nu t)\,(\nu s)\,(\nu r)\,\left(\right.\\
&(\mu X)\,\mathrm{update}\,(n,5)\,.\,\left(\bigodot_{j\in\{1,2\}}\ s\,[4,j]!_u l1a\,\langle \mathrm{proposalNumber}\,(n,4)\rangle\right)\ldots\ = \mathrm{P}_4\\
&\mid\ (\mu X)\,\mathrm{update}\,(n,6)\,.\,\left(\bigodot_{j\in\{2,3\}}\ r\,[4,j]!_u l1a\,\langle \mathrm{proposalNumber}\,(n,5)\rangle\right)\ldots\ = \mathrm{P}_5\\
&\mid\ ((\mu X)\,s\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots\ \mid\ (\mu X)\,r\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots)\ = \mathrm{A}_1\\
&\mid\ ((\mu X)\,s\,[4,2]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots\ \mid\ (\mu X)\,r\,[4,2]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots)\ = \mathrm{A}_2\\
&\mid\ ((\mu X)\,s\,[4,3]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots\ \mid\ (\mu X)\,r\,[4,3]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots)\ = \mathrm{A}_3\\
&\left.\mid \Pi_{1\le k,l\le 4, k\ne l}\ s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4, k\ne l}\ r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2, k\ne l}\ t_{k\to l} : []\right)
\end{aligned}$$

Note that each process is shortened to only show the next few steps instead of the entire process.

The Happy Path

After applying the updates in P_4 and P_5 the first inter-process communication can take place. In this case P_5 communicates with A_2 and A_3. We apply (USend) and (UGet) twice to send $\mathrm{proposalNumber}\,(6,5) = 10$ to A_2 and A_3.

$\longmapsto^* (\nu t)\,(\nu s)\,(\nu r)\,($
$(\mu X)\,s\,[4,1]!_u l1a\,\langle \text{proposalNumber}\,(5,4)\rangle\,.s\,[4,2]!_u l1a\,\langle \text{proposalNumber}\,(5,4)\rangle \ldots = \mathrm{P}_4$
$\mid\ (\mu X)\,r\,[2,4]?_u l1b\,\langle\bot\rangle\,(v_2)\,.r\,[3,4]?_u l1b\,\langle\bot\rangle\,(v_3)\ldots = \mathrm{P}_5$
$\mid\ ((\mu X)\,s\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,r\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots) = \mathrm{A}_1$
$\mid\ ((\mu X)\,s\,[4,2]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,\text{if}\ 10 = \bot\ \text{then}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}\,\text{else}\ \ldots) = \mathrm{A}_2$
$\mid\ ((\mu X)\,s\,[4,3]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,\text{if}\ 10 = \bot\ \text{then}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}\,\text{else}\ \ldots) = \mathrm{A}_3$
$\mid \Pi_{1\le k,l\le 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\ t_{k\to l} : [])$

Since $10 \neq \bot$ both A_2 and A_3 move into their respective else branches.

$= (\nu t)\,(\nu s)\,(\nu r)\,($
$(\mu X)\,s\,[4,1]!_u l1a\,\langle \text{proposalNumber}\,(5,4)\rangle\,.s\,[4,2]!_u l1a\,\langle \text{proposalNumber}\,(5,4)\rangle \ldots = \mathrm{P}_4$
$\mid\ (\mu X)\,r\,[2,4]?_u l1b\,\langle\bot\rangle\,(v_2)\,.r\,[3,4]?_u l1b\,\langle\bot\rangle\,(v_3)\ldots = \mathrm{P}_5$
$\mid\ ((\mu X)\,s\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,r\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots) = \mathrm{A}_1$
$\mid\ ((\mu X)\,s\,[4,2]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,\text{if}\ \text{gt}\,(10,\text{Nothing})\,\text{then}\ \ldots\text{else}\ \ldots) = \mathrm{A}_2$
$\mid\ ((\mu X)\,s\,[4,3]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,\text{if}\ \text{gt}\,(10,\text{Nothing})\,\text{then}\ \ldots\text{else}\ \ldots) = \mathrm{A}_3$
$\mid \Pi_{1\le k,l\le 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\ t_{k\to l} : [])$

Because $\text{gt}\,(10,\text{Nothing})$ returns true, A_2 and A_3 move into their respective then branches. After executing $\text{update}\,(n,10)$, A_2 and A_3 are ready to send their responses to P_5.

$= (\nu t)\,(\nu s)\,(\nu r)\,($
$(\mu X)\,s\,[4,1]!_u l1a\,\langle \text{proposalNumber}\,(5,4)\rangle\,.s\,[4,2]!_u l1a\,\langle \text{proposalNumber}\,(5,4)\rangle \ldots = \mathrm{P}_4$
$\mid\ (\mu X)\,r\,[2,4]?_u l1b\,\langle\bot\rangle\,(v_2)\,.r\,[3,4]?_u l1b\,\langle\bot\rangle\,(v_3)\ldots = \mathrm{P}_5$
$\mid\ ((\mu X)\,s\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,r\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots) = \mathrm{A}_1$
$\mid\ ((\mu X)\,s\,[4,2]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,r\,[2,4]!_u l1b\,\langle\text{Promise}\ \text{Nothing}\rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}) = \mathrm{A}_2$
$\mid\ ((\mu X)\,s\,[4,3]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if}\ \ldots \mid\ (\mu X)\,r\,[3,4]!_u l1b\,\langle\text{Promise}\ \text{Nothing}\rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}) = \mathrm{A}_3$
$\mid \Pi_{1\le k,l\le 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\ t_{k\to l} : [])$

We apply (USend) and (UGet) twice to do just that. Note that we also apply (USkip) to A_1 and evaluate its branches. All three acceptors move into $\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}$.

$\longmapsto^* (\nu t)\,(\nu s)\,(\nu r)\,($
$(\mu X)\,s\,[4,1]!_u l1a\,\langle \text{proposalNumber}\,(5,4)\rangle\,.s\,[4,2]!_u l1a\,\langle \text{proposalNumber}\,(5,4)\rangle \ldots = \mathrm{P}_4$
$\mid\ (\mu X)\,r\,[4,\{2,3\}]!_w \mathit{Accept}.r\,[4,2]!_u l2a\,\langle\text{Proposal}\ 10\ \text{abc}\rangle \ldots = \mathrm{P}_5$
$\mid\ ((\mu X)\ldots \mid\ (\mu X)\,r\,[4,1]?_w \mathit{Accept}\cdots\oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end}) = \mathrm{A}_1$
$\mid\ ((\mu X)\ldots \mid\ (\mu X)\,r\,[4,2]?_w \mathit{Accept}\cdots\oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end}) = \mathrm{A}_2$
$\mid\ ((\mu X)\ldots \mid\ (\mu X)\,r\,[4,3]?_w \mathit{Accept}\cdots\oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end}) = \mathrm{A}_3$
$\mid \Pi_{1\le k,l\le 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\ t_{k\to l} : [])$

P_5 broadcasts its decision *Accept* to A_2 and A_3. By applying (WSel) once, (WBran) twice we obtain:

$$
\begin{array}{l}
\longmapsto^* (\nu t)\,(\nu s)\,(\nu r)\,\big(\\
(\mu X)\,s\,[4,1]!_u l1a\,\langle \mathrm{proposalNumber}\,(5,4)\rangle\,.s\,[4,2]!_u l1a\,\langle \mathrm{proposalNumber}\,(5,4)\rangle \ldots \;= \mathrm{P}_4 \\
\mid\; (\mu X)\,r\,[4,2]!_u l2a\,\langle \mathrm{Proposal}\ 10\ \mathrm{abc}\rangle\,.r\,[4,3]!_u l2a\,\langle \mathrm{Proposal}\ 10\ \mathrm{abc}\rangle \ldots \;= \mathrm{P}_5 \\
\mid\; ((\mu X) \ldots \mid\; (\mu X)\,r\,[4,1]?_w \mathit{Accept} \cdots \oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end}) \;= \mathrm{A}_1 \\
\mid\; ((\mu X) \ldots \mid\; (\mu X)\,r\,[4,2]?_u l2a\,\langle \bot\rangle\,(pr')\,.\,\mathrm{if}\ \ldots) \;= \mathrm{A}_2 \\
\mid\; ((\mu X) \ldots \mid\; (\mu X)\,r\,[4,3]?_u l2a\,\langle \bot\rangle\,(pr')\,.\,\mathrm{if}\ \ldots) \;= \mathrm{A}_3 \\
\mid \Pi_{1\le k,l\le 4,k\neq l}\; s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\; r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\; t_{k\to l} : []\big)
\end{array}
$$

Now P_5 can send its proposal to A_2 and A_3 and terminate. To do so we apply (USend) and (UGet) twice. A_2 and A_3 accept the proposal and the respective subprocesses terminate. Note that we apply (WSkip) in A_1 and terminate that subprocess as well.

$$
\begin{array}{l}
\longmapsto^* (\nu t)\,(\nu s)\,(\nu r)\,\big(\\
(\mu X)\,s\,[4,1]!_u l1a\,\langle \mathrm{proposalNumber}\,(5,4)\rangle\,.s\,[4,2]!_u l1a\,\langle \mathrm{proposalNumber}\,(5,4)\rangle \ldots \;= \mathrm{P}_4 \\
\mid\; (\mu X)\,s\,[4,1]?_u l1a\,\langle \bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots \;= \mathrm{A}_1 \\
\mid\; (\mu X)\,s\,[4,2]?_u l1a\,\langle \bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots \;= \mathrm{A}_2 \\
\mid\; (\mu X)\,s\,[4,3]?_u l1a\,\langle \bot\rangle\,(n')\,.\,\mathrm{if}\ \ldots \;= \mathrm{A}_3 \\
\mid \Pi_{1\le k,l\le 4,k\neq l}\; s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\; r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\; t_{k\to l} : []\big)
\end{array}
$$

At this point the local variables of A_2 and A_3 are $n = 10$ and $pr = \mathrm{Proposal}\ 10\ \mathrm{abc}$. A_1 has not updated its local variables $n = \mathrm{Nothing}$ and $pr = \mathrm{Nothing}$.

Restarting the Algorithm

Next, P_4 sends prepare requests with a proposal number less than 10, which is rejected by A_2. P_4 then decides to restart the algorithm. We apply (USend) and (UGet) twice. We also apply (USkip) once in A_3.

$$
\begin{array}{l}
\longmapsto^* (\nu t)\,(\nu s)\,(\nu r)\,\big(\\
(\mu X)\,s\,[1,4]?_u l1b\,\langle \bot\rangle\,(v_1)\,.s\,[2,4]?_u l1b\,\langle \bot\rangle\,(v_2) \ldots \;= \mathrm{P}_4 \\
\mid\; (\mu X)\,\mathrm{if}\ 5 = \bot\ \mathrm{then}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}\ \mathrm{else}\ \ldots \;= \mathrm{A}_1 \\
\mid\; (\mu X)\,\mathrm{if}\ 5 = \bot\ \mathrm{then}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}\ \mathrm{else}\ \ldots \;= \mathrm{A}_2 \\
\mid\; (\mu X)\,\mathrm{if}\ \bot = \bot\ \mathrm{then}\ \mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}\ \mathrm{else}\ \ldots \;= \mathrm{A}_3 \\
\mid \Pi_{1\le k,l\le 4,k\neq l}\; s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\; r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\; t_{k\to l} : []\big)
\end{array}
$$

A_3 moves directly to $\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}$ whereas A_1 and A_2 send their responses to P_4 before moving to $\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}}$. A_1 also updates its local variable $n = 5$.

$= (\nu t)\,(\nu s)\,(\nu r)\,($
$(\mu X)\,s\,[1,4]?_u l1b\,\langle\bot\rangle\,(v_1)\,.s\,[2,4]?_u l1b\,\langle\bot\rangle\,(v_2)\ldots = \mathrm{P}_4$
$\mid\ (\mu X)\,s\,[1,4]!_u l1b\,\langle\text{Promise Nothing}\rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} = \mathrm{A}_1$
$\mid\ (\mu X)\,s\,[2,4]!_u l1b\,\langle\text{Nack } 10\rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} = \mathrm{A}_2$
$\mid\ (\mu X)\,r\,[4,3]?_w \mathit{Accept}\cdots\oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end} = \mathrm{A}_3$
$\mid \Pi_{1\leq k,l\leq 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\leq k,l\leq 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\leq k,l\leq 2,k\neq l}\ t_{k\to l} : [])$

Applying (USend) and (UGet) twice and evaluating the branching in P_4 yields:

$\longmapsto^*\ (\nu t)\,(\nu s)\,(\nu r)\,($
$(\mu X)\,s\,[4,\{1,2\}]!_w \mathit{Restart}.X$
$\mid\ (\mu X)\,s\,[4,1]?_w \mathit{Accept}\cdots\oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end}$
$\mid\ (\mu X)\,s\,[4,1]?_w \mathit{Accept}\cdots\oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end}$
$\mid\ (\mu X)\,r\,[4,3]?_w \mathit{Accept}\cdots\oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end}$
$\mid \Pi_{1\leq k,l\leq 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\leq k,l\leq 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\leq k,l\leq 2,k\neq l}\ t_{k\to l} : [])$

P_4 sends its decision to restart the algorithm to A_1 and A_2 by applying (WSel) once and (WBran) twice. A_3 terminates after applying (WSkip).

$\longmapsto^*\ (\nu t)\,(\nu s)\,(\nu r)\,($
$(\mu X)\,s\,[4,1]!_u l1a\,\langle 15\rangle\,.s\,[4,2]!_u l1a\,\langle 15\rangle\ldots = \mathrm{P}_4$
$\mid\ (\mu X)\,s\,[4,1]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if } \ldots = \mathrm{A}_1$
$\mid\ (\mu X)\,s\,[4,2]?_u l1a\,\langle\bot\rangle\,(n')\,.\,\text{if } \ldots = \mathrm{A}_2$
$\mid \Pi_{1\leq k,l\leq 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\leq k,l\leq 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\leq k,l\leq 2,k\neq l}\ t_{k\to l} : [])$

The Happy Path, Again

This time P_4 uses a high enough proposal number so that A_1 and A_2 both promise not to accept any proposal numbered less than that. By applying (USend) and (UGet) and evaluating the branches in the remaining acceptors we arrive at:

$\longmapsto^*\ (\nu t)\,(\nu s)\,(\nu r)\,($
$(\mu X)\,s\,[1,4]?_u l1b\,\langle\bot\rangle\,(v_1)\,.s\,[2,4]?_u l1b\,\langle\bot\rangle\,(v_2)\,.\,\text{if } \ldots = \mathrm{P}_4$
$\mid\ (\mu X)\,s\,[1,4]!_u l1b\,\langle\text{Promise Nothing}\rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} = \mathrm{A}_1$
$\mid\ (\mu X)\,s\,[2,4]!_u l1b\,\langle\text{Promise Proposal } 10\ abc\rangle\,.\,\mathrm{P}^{\mathrm{a}}_{\mathrm{cont}} = \mathrm{A}_2$
$\mid \Pi_{1\leq k,l\leq 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\leq k,l\leq 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\leq k,l\leq 2,k\neq l}\ t_{k\to l} : [])$

Note that, at this point, A_1 and A_2 have updated their respective n to 15.

Because A_2 has already accepted a proposal, it responds to P_4's prepare request with that proposal. Twice more we apply (USend) and (UGet) and evaluate the branch in P_4 to obtain:

$$
\begin{array}{l}
\longmapsto^* (\nu t)\,(\nu s)\,(\nu r)\,\big(\\
(\mu X)\,s\,[4,\{1,2\}]!_w \mathit{Accept}.\ldots = \mathrm{P}_4 \\
\mid\ (\mu X)\,s\,[4,1]?_w \mathit{Accept}\cdots \oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end} = \mathrm{A}_1 \\
\mid\ (\mu X)\,s\,[4,1]?_w \mathit{Accept}\cdots \oplus \mathit{Restart}.X \oplus \mathit{Abort}.\mathit{end} = \mathrm{A}_2 \\
\mid \Pi_{1\le k,l\le 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\ t_{k\to l} : []\big)
\end{array}
$$

P_4 has received enough promises to send its own proposal. The value for that proposal is abc because that is the value of the highest-numbered proposal P_4 received as a response to its prepare request. First, we apply (WSel) and (WBran).

$$
\begin{array}{l}
\longmapsto^* (\nu t)\,(\nu s)\,(\nu r)\,\big(\\
(\mu X)\,s\,[4,1]!_u l2a\,\langle \mathrm{Proposal}\ 15\ \mathrm{abc}\rangle\,.s\,[4,2]!_u l2a\,\langle \mathrm{Proposal}\ 15\ \mathrm{abc}\rangle\,.\mathit{end} = \mathrm{P}_4 \\
\mid\ (\mu X)\,s\,[4,1]?_u l2a\,\langle \bot\rangle\,(pr')\,.\,\mathrm{if}\ \ldots = \mathrm{A}_1 \\
\mid\ (\mu X)\,s\,[4,2]?_u l2a\,\langle \bot\rangle\,(pr')\,.\,\mathrm{if}\ \ldots = \mathrm{A}_2 \\
\mid \Pi_{1\le k,l\le 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\ t_{k\to l} : []\big)
\end{array}
$$

Then we apply (USend) and (UGet) to send the proposal from P_4 to the acceptors. P_4 terminates and the acceptors accept the received proposal and then terminate as well.

$$\longmapsto^* (\nu t)\,(\nu s)\,(\nu r)\,\big(\Pi_{1\le k,l\le 4,k\neq l}\ s_{k\to l} : [] \mid \Pi_{1\le k,l\le 4,k\neq l}\ r_{k\to l} : [] \mid \Pi_{1\le k,l\le 2,k\neq l}\ t_{k\to l} : []\big)$$

Afterwards A_1 and A_2 have $n = 15$ and $pr = \mathrm{Proposal}\ 15\ \mathrm{abc}$ and A_3 has $n = 10$ and $pr = \mathrm{Proposal}\ 10\ \mathrm{abc}$. All acceptors have accepted the value abc.

4 Analysis

Bibliography

[1] Tushar Deepak Chandra and Sam Toueg. “Unreliable Failure Detectors for Reliable Distributed Systems”. In: *J. ACM* 43.2 (Mar. 1996), pp. 225–267. ISSN: 0004-5411. DOI: 10.1145/226643.226647. URL: https://doi.org/10.1145/226643.226647.

[2] George Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design (3rd Edition)*. Addison-Wesley, 2001, p. 452.

[3] Leslie Lamport. “Lower Bounds for Asynchronous Consensus”. In: *Distrib. Comput.* 19.2 (Oct. 2006), pp. 104–125. ISSN: 0178-2770. DOI: 10.1007/s00446-006-0155-x. URL: https://doi.org/10.1007/s00446-006-0155-x.

[4] Leslie Lamport. “Paxos Made Simple”. In: *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (Dec. 2001), pp. 51–58. URL: https://www.microsoft.com/en-us/research/publication/paxos-made-simple/.

[5] K. Peters, U. Nestmann, and C. Wagner. “Fault-Tolerant Multiparty Session Types”. Provided by K. Peters. 2021.

[6] A. Scalas and N. Yoshida. “Multiparty session types, beyond duality”. In: *Journal of Logical and Algebraic Methods in Programming* 97 (2018), pp. 55–84.