PAFAS at work: Comparing the Worst-Case Efficiency of Three Buffer Implementations

F. Corradini, M.R. Di Berardini
Dip.to di Matematica Pura ed Applicata
Università dell'Aquila
flavio@univaq.it

W. Vogler
Institut für Informatik
Universität Augsburg
vogler@informatik.uni-augsburg.de

Abstract

In this paper, we use PAFAS (Process Algebra for Faster Asynchronous Systems) to compare the worst-case efficiency of three bounded-buffer implementations: Fifo, Pipe and Buff. Fifo implements the buffer as a first-in-first-out queue, Pipe implements the buffer as a pipeline queue and Buff implements the buffer as circular queue in an array. We contrast our results with those in [1] and [5] which also aim at comparing the three implementations of the bounded buffer according to some efficiency measure.

1 Introduction

Recently, PAFAS has been proposed as a useful tool for comparing the worst-case efficiency of asynchronous systems [6, 3]. PAFAS is a CCS-like process description language [7] where basic actions are atomic and instantaneous but have associated a time bound interpreted as a maximal time delay for their execution. As discussed in [6, 3], due to these upper time bounds time can be used to evaluate efficiency, but it does not influence functionality (which actions are performed); so compared to CCS, also PAFAS treats the full functionality of asynchronous systems. Processes are compared via a variant of the testing approach developed in [4]. Unlike [4], our tests are not simply test environments but test environments together with a time bound. A process is embedded into the environment (via parallel composition) and satisfies a (timed) test, if success is reached before the time bound in *every* run of the composed system, i.e. even in the worst case. This gives rise to a preorder relation over processes which is naturally an *efficiency preorder*. This efficiency preorder can be characterized as inclusion of some kind of refusal traces; this also provides a decidability result for the preorder for finite-state processes. Furthermore, the preorder is independent of the choice to let time progress in a continuous or discrete way; therefore, we only consider discrete time in this paper. These ideas and results were originally successfully studied within the Petri net formalism [8, 5]. We refer the reader to [3] for more details and results on PAFAS.

This paper shows the applicability of PAFAS to concrete meaningful examples. We consider three different implementations of a bounded buffer and relate them according to the above mentioned efficiency preorder. The three implementations are called Fifo, Pipe and Buff. Fifo is a bounded-length first-in-first-out queue, which one could also consider as the specification of a bounded buffer; Pipe is a sequence of one place buffers connected end to end and Buff is an array used in a circular fashion.

We prove that Fifo and Pipe are unrelated according to our (worst-case) efficiency preorder (unrelated means that the former process is not more efficient than the latter one and vice versa); this is presumably in contrast to expectation, since only in Pipe items have to be transported in several steps from one end to the other. Similarly, one would expect Buff to be faster than Pipe, since the latter needs more such steps, but they also turn out to unrelated. We give good reasons for these results and also prove that Fifo is more efficient than Buff, but not vice versa.

For Buff and Pipe, the same results were obtained in [8], where more or less the same efficiency preorder was defined and studied for Petri nets as specification model instead of a process description language such as PAFAS. This shows that the ideas behind our efficiency preorder are not model-dependent – though, of course, the different models impose a different development.

The same buffers we consider were also contrasted in [1]. Their approach is based on a bisimulation-based preorder; visible actions are regarded as instantaneous and the costs are measured as the number of internal actions. Hence [1] presents an interleaving approach, which disregards the parallel execution of actions. According to this efficiency measure, it has been proven that Fifo is more efficient than Buff and Buff is more efficient than Pipe. Parallel execution of actions is taken into account in the present paper. This is the reason why Pipe is incomparable with the others in our approach.

The rest of the paper is organized as follows. The next section briefly recalls PAFAS, the testing scenario and the alternative characterization in terms of refusal traces. Section 3 provides a description of the three buffer implementations and their operational behaviour, while Section 4 studies their relationships according to the efficiency preorder.

2 PAFAS

In this section we briefly introduce our process description language PAFAS, its operational semantics and a preorder relating processes according to the worst-case efficiency. We refer the reader to [3] for more details.

2.1 Timed Processes and Tests

The specification language we consider is a CCS-like language (with *TCSP*-like parallel composition). We use the following notation:

- $\mathbb{A}$ is an infinite set of actions with the special action ω, which is reserved for observers (test processes) in the testing scenario to signal success of a test. The additional action τ represents internal activity that is unobservable for other components. We define $\mathbb{A}_\tau = \mathbb{A} \cup \{\tau\}$. Elements of $\mathbb{A}$ are denoted by $a, b, c, \ldots$ and those of $\mathbb{A}_\tau$ are denoted by $\alpha, \beta, \ldots$. Actions in $\mathbb{A}_\tau$ can let time 1 pass before their execution, i.e. 1 is their maximum delay. After that time, they become *urgent* actions. The set of urgent actions is denoted by $\underline{\mathbb{A}}_\tau = \{\underline{a} \mid a \in \mathbb{A}\} \cup \{\underline{\tau}\}$ and is ranged over by $\underline{\alpha}, \underline{\beta}, \ldots$

- χ is the set of process variables, used for recursive definitions. Elements of χ are denoted by $x, y, z, \ldots$.

- $\Phi : \mathbb{A}_\tau \to \mathbb{A}_\tau$ is a function such that the set $\{\alpha \in \mathbb{A}_\tau \mid \emptyset \neq \Phi^{-1}(\alpha) \neq \{\alpha\}\}$ is finite, $\Phi^{-1}(\omega) \subseteq \{\omega\}$ and $\Phi(\tau) = \tau$; then Φ is a *general relabelling function*. As shown in [3], general relabelling functions subsume the classically distinguished operations relabelling and hiding: P/A, where the actions in A are made internal, is the same as $P[\Phi_A]$, where the relabelling function Φ_A is defined by $\Phi_A(\alpha) = \tau$ if $\alpha \in A$ and $\Phi_A(\alpha) = \alpha$ if $\alpha \notin A$.

Definition 2.1 (*Timed and Initial Processes*)
The set $\mathbb{P}$ of *(discretely timed) processes* is the set of closed (i.e., without free variables) and guarded (i.e., variable x in a $\mu x.P$ only appears within the scope of an $\alpha.()$-prefix with $\alpha \in \mathbb{A}_\tau$) terms generated by the following grammar:

$$P ::= 0 \mid \alpha.P \mid \underline{\alpha}.P \mid P + P \mid P \|_A P \mid P[\Phi] \mid x \mid \mu x.P$$

where $x \in \chi$, $\alpha \in \mathbb{A}_\tau$, Φ a general relabelling function and $A \subseteq \mathbb{A}$ possibly infinite. $\mathbb{P}_1$ is the set of so called *initial processes*, i.e. processes where all actions are from set $\mathbb{A}_\tau$. This is a significant subset of $\mathbb{P}$ since it corresponds to ordinary CCS-like processes.

0 is the Nil-process, which cannot perform any action, but may let time pass without limit; a trailing 0 will often be omitted, so e.g. $a.b + c$ abbreviates $a.b.0 + c.0$. $\alpha.P$ and $\underline{\alpha}.P$ is (action-) prefixing, known from CCS. Process $\alpha.P$ performs α with a *maximal* delay of 1; hence, it can perform α immediately or it can idle for time 1 and become $\underline{\alpha}.P$. In the latter case, the idle-time has elapsed; hence α must either occur or be deactivated (in a choice-context) before time may pass further – unless it has to wait for synchronization with another component (in case $\alpha \neq \tau$). This means that our processes are *patient*: As a stand-alone process, $\underline{a}.P$ has no reason to wait; but as a component in $(\underline{a}.P)\|_{\{a\}}(a.Q)$, it has to wait for synchronization on a and this can take up to time 1, since component $a.Q$ may idle this long. $P_1 + P_2$ models the choice (sum) of two conflicting processes P_1 and P_2. $P_1\|_A P_2$ is the parallel composition of two processes P_1 and P_2 that run in parallel and have to synchronize on all actions from A; this synchronization discipline is inspired from *TCSP*. $P[\Phi]$ behaves as P but with the actions changed according to Φ. $\mu x.P$ models a recursive definition; in the examples, we will define processes by recursive equations instead.

Now the purely functional behaviour of processes (i.e. which actions they can perform) is given by the following operational semantics.

Definition 2.2 (*Operational semantics of functional behaviour*) The following SOS-rules define the transition relations $\xrightarrow{\alpha} \subseteq (\mathbb{P} \times \mathbb{P})$ for $\alpha \in \mathbb{A}_\tau$, the *action transitions*.

As usual, we write $P \xrightarrow{\alpha} P'$ if $(P, P') \in \xrightarrow{\alpha}$ and $P \xrightarrow{\alpha}$ if there exists a $P' \in \mathbb{P}$ such that $(P, P') \in \xrightarrow{\alpha}$, and similar conventions will apply later on.

$$\text{Pref}_{a1}\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad \text{Pref}_{a2}\ \frac{}{\underline{\alpha}.P \xrightarrow{\alpha} P}$$

$$\text{Par}_{a1}\ \frac{\alpha \notin A,\ P_1 \xrightarrow{\alpha} P_1'}{P_1\|_A P_2 \xrightarrow{\alpha} P_1'\|_A P_2} \qquad \text{Par}_{a2}\ \frac{\alpha \in A,\ P_1 \xrightarrow{\alpha} P_1',\ P_2 \xrightarrow{\alpha} P_2'}{P_1\|_A P_2 \xrightarrow{\alpha} P_1'\|_A P_2'}$$

$$\text{Sum}_a\ \frac{P_1 \xrightarrow{\alpha} P_1'}{P_1 + P_2 \xrightarrow{\alpha} P_1'} \qquad \text{Rel}_a\ \frac{P \xrightarrow{\alpha} P'}{P[\Phi] \xrightarrow{\Phi(\alpha)} P'[\Phi]} \qquad \text{Rec}_a\ \frac{P\{\mu x.P/x\} \xrightarrow{\alpha} P'}{\mu x.P \xrightarrow{\alpha} P'}$$

Additionally, there are symmetric rules for Par_{a1} and Sum_a for actions of P_2. Finally, $\mathcal{A}(P) = \{\alpha \in \mathbb{A}_\tau \mid P \xrightarrow{\alpha}\}$ is the set of *activated actions* of P.

Except for Pref_{a2}, these rules are standard.[1] Pref_{a1} and Pref_{a2} allow an activated action to occur (just as e.g. in CCS), and it makes *no* difference whether the action is urgent or not. Additionally, passage of time will never deactivate actions or activate new ones, and we capture all behaviour that is possible in the standard CCS-like setting without time.

The set of activated actions $\mathcal{A}(P)$ of a process P describes its immediate functional behaviour. It records only actions, not the possibly various timer values associated with the same action in a process, and is finite as proven in [3].

As a first step to define timed behaviour, we now give operational rules for the passage of *discrete* 'wait-time': all components of a system participate in a global time step, and this passage of time is recorded for locally activated actions by decreasing their delay in rule Pref_{d1}. Note that in 2.3, time passes disregarding elapsed delays; as in the example above, this might be necessary for a component when waiting for a synchronization partner, explaining the name 'wait-time'.

[1] [3] uses a different rule Rec_a which is equivalent due to guardedness.

Definition 2.3 (*Operational semantics for wait-time*) Via the following SOS-rules, a relation $\overset{1}{\rightsquigarrow} \subseteq (\mathbb{P} \times \mathbb{P})$ is defined:

$\text{Nil}_d \ \dfrac{}{0 \overset{1}{\rightsquigarrow} 0}$ $\qquad$ $\text{Pref}_{d1} \ \dfrac{}{\alpha.P \overset{1}{\rightsquigarrow} \underline{\alpha}.P}$ $\qquad$ $\text{Pref}_{d2} \ \dfrac{}{\underline{\alpha}.P \overset{1}{\rightsquigarrow} \underline{\alpha}.P}$

$\text{Sum}_d \ \dfrac{P_1 \overset{1}{\rightsquigarrow} P_1',\ P_2 \overset{1}{\rightsquigarrow} P_2'}{P_1 + P_2 \overset{1}{\rightsquigarrow} P_1' + P_2'}$ $\qquad$ $\text{Par}_d \ \dfrac{P_1 \overset{1}{\rightsquigarrow} P_1',\ P_2 \overset{1}{\rightsquigarrow} P_2'}{P_1 \|_A P_2 \overset{1}{\rightsquigarrow} P_1' \|_A P_2'}$

$\text{Rel}_d \ \dfrac{P \overset{1}{\rightsquigarrow}_d P'}{P[\Phi] \overset{1}{\rightsquigarrow} P'[\Phi]}$ $\qquad$ $\text{Rec}_d \ \dfrac{P\{\mu x.P/x\} \overset{1}{\rightsquigarrow} P'}{\mu x.P \overset{1}{\rightsquigarrow} P'}$

The operational semantics of wait-time allows general processes to wait forever, but our intention was that an urgent action has to occur or be disabled (– unless it has to wait for a synchronization partner). We will enforce this using an auxiliary function that calculates for a given action α its residual time $\mathcal{R}(\alpha, P)$ in a process P, i.e. the time until it becomes urgent (in case it is activated at all).

Definition 2.4 (*Residual time of actions and processes*) The *residual time* $\mathcal{R}(\alpha, P)$ of an action $\alpha \in \mathbb{A}_\tau$ in a process $P \in \mathbb{P}$ is defined by:

Nil: $\quad \mathcal{R}(\alpha, 0) = 1$

Pref: $\quad \mathcal{R}(\alpha, \gamma.P) = \begin{cases} 0 & \text{if } \gamma = \underline{\alpha} \\ 1 & \text{otherwise} \end{cases}$

Sum: $\quad \mathcal{R}(\alpha, P_1 + P_2) = \min(\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2))$

Par: $\quad \mathcal{R}(\alpha, P_1 \|_A P_2) = \begin{cases} \max(\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2)) & \text{if } \alpha \in A \\ \min(\mathcal{R}(\alpha, P_1), \mathcal{R}(\alpha, P_2)) & \text{if } \alpha \notin A \end{cases}$

Rel: $\quad \mathcal{R}(\alpha, P[\Phi]) = \min\{\mathcal{R}(\beta, P) \mid \beta \in \Phi^{-1}(\alpha)\}$

Rec: $\quad \mathcal{R}(\alpha, \mu x.P) = \mathcal{R}(\alpha, P\{\mu x.P/x\})$

In this definition, we put $\min \emptyset := 1$ (for case Rel). The *residual time* of a process $P \in \mathbb{P}$ is $\mathcal{R}(P) = \min\{\mathcal{R}(\alpha, P) \mid \alpha \in \mathcal{A}(P)\}$.

As shown in [3], $\mathcal{R}(P)$ exists for each process P, and, hence, the residual time is well-defined in all cases. The most important case is the Par-case which realizes the desired behaviour of waiting in a parallel composition: if P_1 and P_2 have to synchronize on α, then the residual time of α in $P_1 \|_A P_2$ is determined by the 'slower' component with larger residual time; if P_1 and P_2 do not have to synchronize on α, the 'faster' component determines the maximal possible delay of α in $P_1 \|_A P_2$.

The effect of waiting on the residual time of activated actions is described by the following lemma: if time advances by 1, then the residual time of an activated action is decreased by the same amount, i.e. it is zero afterwards. This behaviour is realized locally by rule Pref_c of Definition 2.3.

Lemma 2.5 For processes $P, P' \in \mathbb{P}$ let $P \overset{1}{\rightsquigarrow} P'$. Then:

1. for all $\alpha \in \mathcal{A}(P) = \mathcal{A}(P')$ we have $\mathcal{R}(\alpha, P') = 0$;
2. $\mathcal{R}(\alpha, P) \neq 1$ implies $\alpha \in \mathcal{A}(P)$.

The residual time is the time a stand-alone process can idle; thus, we can use it to restrict wait-time to the timed behaviour we had in mind originally and which we call 'idle-time'.

Definition 2.6 (*Passage of idle-time*) For $P, P' \in \mathbb{P}$ we write $P \xrightarrow{1} P'$ if $P \overset{1}{\rightsquigarrow} P'$ and $\mathcal{R}(P) = 1$.

Both, purely functional and timed behaviour of processes will now be combined in the (discrete) language of processes. As usual, we will abstract from internal behaviour; but note that internal actions gain some 'visibility' in timed behaviour, since their presence possibly allows to pass more time in between the occurrence of visible actions.

Definition 2.7 (*Language of processes*)

Let $P, P' \in \mathbb{P}$ be processes. We extend the transition relation $P \xrightarrow{\mu} P'$ for $\mu \in \mathbb{A}_\tau$ or $\mu = 1$ to sequences w and write $P \xrightarrow{w} P'$ if $P \equiv P'$ and $w = \varepsilon$ (the empty sequence) or there exist $Q \in \mathbb{P}$ and $\mu \in \mathbb{A}_\tau \cup \{1\}$ such that $P \xrightarrow{\mu} Q \xrightarrow{w'} P'$ and $w = \mu w'$.

For a sequence $w \in (\mathbb{A}_\tau \cup \{1\})^*$, let w/τ be the sequence w with all τ's removed, and let the *duration* $\zeta(w)$ of w be the number of time steps in w; note that $\zeta(w/\tau) = \zeta(w)$. We write $P \overset{v}{\Rightarrow} P'$, if $P \xrightarrow{w} P'$ and $v = w/\tau$. Now we define $\mathsf{DL}(P) = \{w \mid P \overset{w}{\Rightarrow}\}$ to be the *(discretely timed) language*, containing the *(discrete) traces* of P.

Based on the language of processes, we are now ready to define timed testing and to relate processes w.r.t. their efficiency, thereby defining an *efficiency preorder*:

Definition 2.8 (*Timed tests*) A process $P \in \mathbb{P}$ is *testable* if ω does not occur in P. Any *initial* process $O \in \mathbb{P}_1$ may serve as a *test process (observer)*. We write $\|$ for $\|_{\mathbb{A}\setminus\{\omega\}}$.

A *timed test* is a pair (O, R), where O is a test process and $R \in \mathbb{N}_0^+$ is the *time bound*. A testable process P *d-satisfies* a timed test (P *must*$_d$ (O, R)), if each $w \in \mathsf{DL}(\tau.P \| O)$ with $\zeta(w) > R$ contains some ω.

For testable processes P and Q, we call P a *faster implementation* of Q or *faster than* Q, written $P \sqsupseteq Q$, if P d-satisfies all timed tests that Q d-satisfies.

For technical reasons (precongruence results) we have to consider $\tau.P \| O$ (which is shorthand for $(\tau.P) \|_{\mathbb{A}\setminus\{\omega\}} O$) instead of $P \| O$ as usual (e.g. [4]). From an intuitive point of view, the additional τ-prefix represents some internal setup activity before the actual test begins.

Runs with duration less than R may not contain all actions that occur up to time R; hence we only consider runs with a duration greater than the time bound R for test satisfaction. The operational idea behind this is that – when performing a test – one should certainly wait until time R is up before declaring the test a failure. By definition, $P \sqsupseteq Q$ means that P is functionally a refinement of Q, since it is satisfactory for at least as many test processes as Q, and that it is an improvement timewise, since it d-satisfies test processes at least as fast as Q.

Obviously, it is impossible to apply the definition directly, since there are already countably many time bounds and, hence, timed tests to apply. In the next section, we introduce a kind of refusal traces which provide a means to check $P \sqsupseteq Q$ for given testable P and Q. These refusal traces give quite a detailed account of the timed behaviour of processes; this is quite surprising, since we are in an asynchronous setting, where tests should have little temporal control over the tested system, and we only use initial processes as tests. In fact, general processes as tests would not "see more", cf. [3].

2.2 Characterization

We first modify the SOS-rules for wait-time as follows: we only allow unit time steps and record at each time step a so-called *refusal set* X of actions which are *not* waiting; i.e. these actions are *not* urgent, they do not have to be performed and can be refused at this moment. Note that in contrast to wait-time we now prohibit passage of time if an urgent τ can be performed.

Definition 2.9 (*SOS-rules for refusal of actions, refusal traces*) The following SOS-rules define $\xrightarrow{X}_r \subseteq (\mathbb{P} \times \mathbb{P})$, where $X, X_i \subseteq \mathbb{A}$:

$$\text{Nil}_r \ \frac{}{0 \xrightarrow{X}_r 0} \qquad \text{Pref}_{r1} \ \frac{}{\alpha.P \xrightarrow{X}_r \underline{\alpha}.P} \qquad \text{Pref}_{r2} \ \frac{\alpha \notin X \cup \{\tau\}}{\underline{\alpha}.P \xrightarrow{X}_r \underline{\alpha}.P}$$

$$\text{Par}_r \ \frac{\forall_{i=1,2}\ P_i \xrightarrow{X_i}_r P_i',\ X \subseteq (A \cap \bigcup_{i=1,2} X_i) \cup ((\bigcap_{i=1,2} X_i) \setminus A)}{P_1 \|_A P_2 \xrightarrow{X}_r P_1' \|_A P_2'}$$

$$\text{Sum}_r \ \frac{\forall_{i=1,2}\ P_i \xrightarrow{X}_r P_i'}{P_1 + P_2 \xrightarrow{X}_r P_1' + P_2'} \qquad \text{Rel}_r \ \frac{P \xrightarrow{\Phi^{-1}(X \cup \{\tau\}) \setminus \{\tau\}}_r P'}{P[\Phi] \xrightarrow{X}_r P'[\Phi]} \qquad \text{Rec}_r \ \frac{P\{\mu x.P/x\} \xrightarrow{X}_r P'}{\mu x.P \xrightarrow{X}_r P'}$$

When $P \xrightarrow{X}_r P'$, we call this a *time step.*

For processes $P, P' \in \mathbb{P}$, we write $P \xrightarrow{\mu}_r P'$, if either $\mu = \alpha \in \mathbb{A}_\tau$ and $P \xrightarrow{\alpha} P'$, or $\mu = X \subseteq \mathbb{A}$ and $P \xrightarrow{X}_r P'$. For sequences w, we define $P \xrightarrow{w}_r P'$ and $P \overset{w}{\Rightarrow}_r P'$ analogously to Definition 2.7.

$\mathsf{RT}(P) = \{w \mid P \overset{w}{\Rightarrow}_r\}$ is the set of *refusal traces* of P. We write $P \leq_r Q$ if $\mathsf{RT}(P) \subseteq \mathsf{RT}(Q)$.

By Proposition 2.10.1 below, the set of possible refusal sets at a time step is downward closed w.r.t. set inclusion, and by .3, non-activated actions can always be refused. Hence, only the refusal of activated actions is relevant to determine the time steps of a process. Proposition 2.10.4 links time steps to unit-time-waiting, unit-time-idling resp.

Proposition 2.10 Let $P, Q, R \in \mathbb{P}$ be processes and let $X, X' \subseteq \mathbb{A}$.

1. If $P \xrightarrow{X}_r Q$ and $X' \subseteq X$, then $P \xrightarrow{X'}_r Q$.
2. If $P \xrightarrow{X}_r Q$ and $P \xrightarrow{X'}_r R$, then $Q \equiv R$.
3. If $P \xrightarrow{X}_r Q$ and $X' \cap \mathcal{A}(P) = \emptyset$, then $P \xrightarrow{X \cup X'}_r Q$.
4. $P \xrightarrow{X}_r Q$ if and only if $P \overset{1}{\rightsquigarrow} Q$ and $\forall_{\alpha \in X \cup \{\tau\}}\ \mathcal{R}(\alpha, P) = 1$,
   in particular $P \xrightarrow{\mathbb{A}}_r Q$ if and only if $P \xrightarrow{1} Q$.

The efficiency preorder is characterized by refusal-trace-inclusion (see [3] for the proof).

Theorem 2.11 (*Characterization of the testing preorder*) Let P_1, P_2 be testable processes. Then $P_1 \sqsupseteq P_2$ if and only if $P_1 \leq_r P_2$.

If P is not faster than Q, i.e. $P \not\sqsupseteq Q$, then there is a refusal trace of P that is not one of Q. This is a witness of slow behaviour of P; it is a diagnostic information that tells us why P is not faster. If P and Q are finite-state, inclusion of refusal traces can be checked automatically; a respective tool, FastAsy, has been developed for a Petri net setting [2], and we plan to adapt this for PAFAS. In case that P is not faster, FastAsy presents a respective refusal trace; this can be used to improve P – and in practice, it can also help to find errors that can occur when formalizing an intuitive idea as a PAFAS-process.

3 Description of the Three Buffers

In this section we describe three implementations of a buffer of capacity $N+2$, where N is a fixed positive natural number. The buffer receives and stores values from the set $\mathrm{V} = \{0,1\}$. We use the following notation: Strings are denoted s, t,... and $|s|$ denotes the length of s; thus, $|s| = 0$ means $s = \varepsilon$. V^k denotes the set of strings in V^* of length k while $\mathrm{V}_k = \bigcup_{i=0}^{k} \mathrm{V}^i$ denotes the set of strings of length at least k.

In the following, we will use the notation $\sum_{e\in \mathrm{V}} P(e)$, where the "value-variable" e appears in $P(e)$; the notation stands for $P(0) + P(1)$, where $P(0)$ is obtained by substituting 0 for e in a way that will be obvious, and similarly for $P(1)$.

We also extend V to include values $\underline{0}, \underline{1}$ and put $D = \{0,1,\underline{0},\underline{1}\}$ and $D_\perp = D \cup \{\perp, \underline{\perp}\}$. $\perp$ is a special value which will denote the absence of values. We assume the properties $\underline{\underline{0}} = \underline{0}$ and $\underline{\underline{1}} = \underline{1}$. The extension of _ to strings is as expected.

3.1 Buffer "Fifo"

Buffer Fifo directly implements a first-in-first-out queue of capacity $N+2$; it has no overhead in the form of internal actions, and it is purely sequential. The state is denoted by the string contained in Fifo; thus, the state space is given by V_{N+2}. We now give a formal definition and then prove some important properties. The software architecture of Fifo is described in Fig. 1. The following formal definition of $\mathsf{Fifo}(s)$ corresponds to the one in [1], while the definition of $\underline{\mathsf{Fifo}}(s)$ is indigenous to our approach since it describes the process $\mathsf{Fifo}(s)$ evolves to after a time-step transition. (This applies analogously for the other two implementations below.)

Figure 1: The Software Architecture for Fifo

Definition 3.1 (*Buffer* Fifo) Let $s \in \mathrm{V}_{N+2}$ and $d \in \mathrm{V}$. We define $\mathsf{Fifo} \equiv \mathsf{Fifo}(\varepsilon)$ where $\mathsf{Fifo}(s)$ is defined by the following recursive equations:

1. if $|s| = 0$ (and $s = \varepsilon$), then $\mathsf{Fifo}(s) \equiv \sum_{e\in \mathrm{V}} in(e).\mathsf{Fifo}(e)$
2. if $0 < |s| < N+2$ and $s = ds'$, then $\mathsf{Fifo}(s) \equiv out(d).\mathsf{Fifo}(s') + \sum_{e\in \mathrm{V}} in(e).\mathsf{Fifo}(se)$
3. if $|s| = N+2$ and $s = ds'$, then $\mathsf{Fifo}(s) \equiv out(d).\mathsf{Fifo}(s')$

The target state for a time step of $\mathsf{Fifo}(s)$ is denoted by $\underline{\mathsf{Fifo}}(s)$ and is defined by the following recursive definitions:

1. if $|s| = 0$ (and $s = \varepsilon$), then $\underline{\mathsf{Fifo}}(s) \equiv \sum_{e\in \mathrm{V}} \underline{in}(e).\mathsf{Fifo}(e)$
2. if $0 < |s| < N+2$ and $s = ds'$, then $\underline{\mathsf{Fifo}}(s) \equiv \underline{out}(d).\mathsf{Fifo}(s') + \sum_{e\in \mathrm{V}} \underline{in}(e).\mathsf{Fifo}(se)$
3. if $|s| = N+2$ and $s = ds'$, then $\underline{\mathsf{Fifo}}(s) \equiv \underline{out}(d).\mathsf{Fifo}(s')$

Some properties that will be used to relate the current implementation of the buffer with the other ones are listed below. They indicate how $\mathsf{Fifo}(s)$ and $\underline{\mathsf{Fifo}}(s)$ evolve after $\xrightarrow{\alpha}_r$- and $\xrightarrow{X}_r$-transitions.

Proposition 3.2 Assume $s \in \mathrm{V}_{N+2}$ and $d \in \mathrm{V}$. Then all transitions of the processes $\mathsf{Fifo}(s)$ and $\underline{\mathsf{Fifo}}(s)$ are the following:

1. $|s| < N+2$ implies $\mathsf{Fifo}(s) \xrightarrow{in(d)}_r \mathsf{Fifo}(sd)$;
2. $|s| > 0$ and $s = ds'$ implies $\mathsf{Fifo}(s) \xrightarrow{out(d)}_r \mathsf{Fifo}(s')$;
3. $\mathsf{Fifo}(s) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(s)$, for every $X \subseteq \mathbb{A}$;
4. $|s| < N+2$ implies $\underline{\mathsf{Fifo}}(s) \xrightarrow{in(d)}_r \mathsf{Fifo}(sd)$;
5. $|s| > 0$ and $s = ds'$ implies $\underline{\mathsf{Fifo}}(s) \xrightarrow{out(d)}_r \mathsf{Fifo}(s')$;
6. $|s| = 0$ (i.e. $s = \varepsilon$) implies $\underline{\mathsf{Fifo}}(s) \xrightarrow{Y}_r \underline{\mathsf{Fifo}}(s)$, for every $Y \subseteq \mathbb{A}\backslash\{in(0), in(1)\}$;
7. $0 < |s| < N+2$ and $s = ds'$ implies $\underline{\mathsf{Fifo}}(s) \xrightarrow{Y}_r \underline{\mathsf{Fifo}}(s)$, for every $Y \subseteq \mathbb{A}\backslash\{out(d), in(0), in(1)\}$;
8. $|s| = N+2$ and $s = ds'$ implies $\underline{\mathsf{Fifo}}(s) \xrightarrow{Y}_r \underline{\mathsf{Fifo}}(s)$, for every $Y \subseteq \mathbb{A}\backslash\{out(d)\}$.

Proof: We just prove items 1., 3. and 6. The others follow similarly.

1. Assume $|s| < N+2$ and distinguish two cases:

   1.1 $|s| = 0$. Then: $\mathsf{Fifo}(s) \equiv in(0).\mathsf{Fifo}(0) + in(1).\mathsf{Fifo}(1) \xrightarrow{in(d)}_r \mathsf{Fifo}(d)$.

   1.2 $0 < |s| < N+2$ and $s = ds'$. Then:
   $\mathsf{Fifo}(s) \equiv out(d).\mathsf{Fifo}(s') + in(0).\mathsf{Fifo}(0) + in(1).\mathsf{Fifo}(1) \xrightarrow{in(d)}_r \mathsf{Fifo}(sd)$.

   In both cases $\mathsf{Fifo}(s) \xrightarrow{in(d)}_r \mathsf{Fifo}(sd)$.

3. The operational rule defining $\overset{1}{\rightsquigarrow}$ and Definition 3.1 imply $\mathsf{Fifo}(s) \overset{1}{\rightsquigarrow} \underline{\mathsf{Fifo}}(s)$. Moreover, $\mathsf{Fifo}(s)$ is an initial process and, hence, for every $\alpha \in \mathbb{A}_\tau$ we have $\mathcal{R}(\alpha, \mathsf{Fifo}(s)) = 1$. Then, by Proposition 2.10.4, $\mathsf{Fifo}(s) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(s)$, for every $X \subseteq \mathbb{A}$.

6. Similarly to the previous case, $\underline{\mathsf{Fifo}}(\varepsilon) \rightsquigarrow \underline{\mathsf{Fifo}}(\varepsilon)$. By Lemma 2.5.2, $\alpha \in \mathbb{A}_\tau\backslash\{in(0), in(1)\}$ implies $\mathcal{R}(\alpha, \underline{\mathsf{Fifo}}(\varepsilon)) = 1$. Then, by Proposition 2.10.4, $\mathsf{Fifo}(s) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(s)$, for every $X \subseteq \mathbb{A}\backslash\{in(0), in(1)\}$.

□

3.2 Buffer "Pipe"

A buffer can also be seen as the "concatenation" of cells, each of them containing at most one value. A cell is an input-output device. It is defined as follows:

Definition 3.3 (*Cells*)

1. $\mathsf{C}(\bot) \equiv \sum_{e \in \mathrm{V}} in(e).\mathsf{C}(e)$ denotes the *empty cell*. It can only input either value 0 or 1;
2. $\mathsf{C}(d) \equiv out(d).\mathsf{C}(\bot)$ denotes a cell containing the value $d \in \mathrm{V}$. It can only output value d;

3. $\mathsf{C}(\underline{\perp}) \equiv \sum_{e \in \mathrm{V}} \underline{in}(e).\mathsf{C}(e)$ denotes the empty cell after a time-step;

4. $\mathsf{C}(\underline{d}) \equiv \underline{out}(d).\mathsf{C}(\perp)$ denotes the cell containing a value d after a time-step.

The special value $\perp$, contained in the empty cell, denotes the absence of values in the cell. A buffer of capacity $N+2$ can be obtained by connecting $N+2$ cells end to end and relabelling communication channels appropriately. Every value pushed into the buffer crosses every cell before reaching the last one from where the value can be popped. The actions $\delta_j(d)$, $d \in \mathrm{V}$, denote the sending of value d from the $j+1$th to the jth cell. The software architecture of Pipe is described in Fig 2.

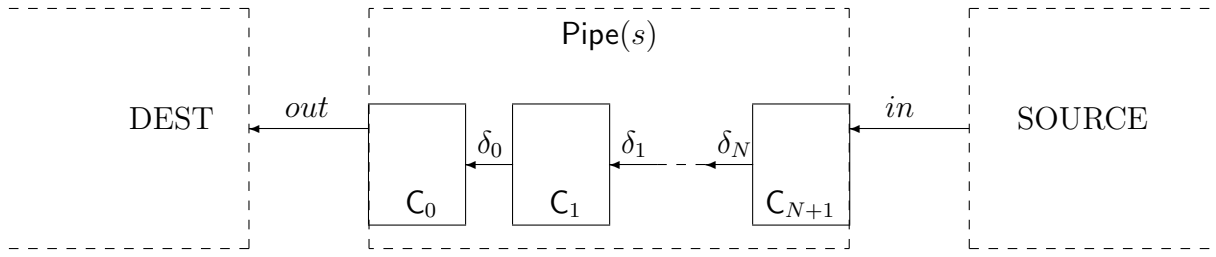

Figure 2: The Software Architecture for Pipe

Definition 3.4 (*Buffer* Pipe) Let $x \in D_\perp$ and $j = 0, \ldots, N+1$. The j-th cell of Pipe is defined by $\mathsf{C}_j(x) \equiv \mathsf{C}(x)[\Phi_j]$, where the various relabelling functions Φ_j are defined as:

$$\Phi_0(\alpha) = \begin{cases} \delta_0(0) & \text{if} \quad \alpha = in(0) \\ \delta_0(1) & \text{if} \quad \alpha = in(1) \\ \alpha & \text{otherwise} \end{cases} \qquad \Phi_{N+1}(\alpha) = \begin{cases} \delta_N(0) & \text{if} \quad \alpha = out(0) \\ \delta_N(1) & \text{if} \quad \alpha = out(1) \\ \alpha & \text{otherwise} \end{cases}$$

and for $j = 1, \ldots, N$,

$$\Phi_j(\alpha) = \begin{cases} \delta_j(0) & \text{if} \quad \alpha = in(0) \\ \delta_j(1) & \text{if} \quad \alpha = in(1) \\ \delta_{j-1}(0) & \text{if} \quad \alpha = out(0) \\ \delta_{j-1}(1) & \text{if} \quad \alpha = out(1) \\ \alpha & \text{otherwise} \end{cases}$$

Now, let $A_j = \{\delta_j(0), \delta_j(1)\}$, for every $j = 0, \ldots, N$, and $A = \bigcup_{j=0}^{N} A_j$. For any string $s = s_0 \ldots s_{N+1} \in D_\perp^{N+2}$, define

$$I(s) \equiv \mathsf{C}_0(s_0) \parallel_{A_0} \mathsf{C}_1(s_1) \parallel_{A_1} \ldots \parallel_{A_N} \mathsf{C}_{N+1}(s_{N+1})$$

Observe that in this parallel composition associativity holds since the actions in A_j can only be performed by $C_j(s_j)$ and $C_{j+1}(s_{j+1})$.

Finally, we define Pipe as $\mathsf{Pipe}(\perp^{N+2})$ where $\mathsf{Pipe}(s)$ in turn is defined as $I(s)/A$, i.e. it is obtained by abstracting from internal details. Recall that $I(s)/A \equiv I(s)[\Phi_A]$, where the relabelling function Φ_A is defined as $\Phi_A(\alpha) = \tau$ if $\alpha \in A$ and $\Phi_A(\alpha) = \alpha$ if $\alpha \notin A$.

To describe the behaviour of Pipe , we start by considering transitions performed by cells.

Proposition 3.5 Assume $d \in \mathrm{V}$. All action transitions of the processes $\mathsf{C}_j(x)$ are the following:

1. $x \in \{\bot, \underline{\bot}\}$ implies $\mathsf{C}_j(x) \xrightarrow{\delta_j(d)}_r \mathsf{C}_j(d)$, for $j = 0, \ldots, N$.

2. $x \in \{\bot, \underline{\bot}\}$ implies $\mathsf{C}_{N+1}(x) \xrightarrow{in(d)}_r \mathsf{C}_{N+1}(d)$.

3. $x \in \{d, \underline{d}\}$ implies $\mathsf{C}_j(x) \xrightarrow{\delta_{j-1}(d)}_r \mathsf{C}_j(\bot)$, for $j = 1, \ldots, N+1$.

4. $x \in \{d, \underline{d}\}$ implies $\mathsf{C}_0(x) \xrightarrow{out(d)}_r \mathsf{C}_0(\bot)$.

Proof: The proof is similar to the previous one of Proposition 3.2 and follows directly from the definition of cell and $\xrightarrow{\mu}_r$. □

Next, we state the transitional results of process $I(s)$ for a string $s \in D_\bot^{N+2}$.

Proposition 3.6 Assume $s = s_0 \ldots s_{N+1} \in D_\bot^{N+2}$ and $d \in$ V. All action transitions of the process $I(s)$ are the following:

1. $s_{N+1} \in \{\bot, \underline{\bot}\}$ implies $I(s) \xrightarrow{in(d)}_r I(s'd)$, where $s' = s_0 \ldots s_N$;

2. $s_0 \in \{d, \underline{d}\}$ implies $I(s) \xrightarrow{out(d)}_r I(\bot\ s')$, where $s' = s_1 \ldots s_{N+1}$;

3. if there exists $j \in \{0, \ldots, N\}$ such that $s_j \in \{\bot, \underline{\bot}\}$ and $s_{j+1} \in \{d, \underline{d}\}$ then $I(s) \xrightarrow{\delta_j(d)}_r I(s'd \perp s'')$, where $s' = s_0 \ldots s_{j-1}$ (which is ε for $j = 0$) and $s'' = s_{j+2} \ldots s_{N+1}$ (which is ε for $j = N$).

Proof:

1. By Proposition 3.5, if $s_{N+1} \in \{\bot, \underline{\bot}\}$, $\mathsf{C}_{N+1}(s_{N+1}) \xrightarrow{in(d)}_r \mathsf{C}_{N+1}(d)$. Since $in(d) \notin A_N$, we also have $I(s) \equiv \mathsf{C}_0(s_0) \|_{A_0} \ldots \|_{A_{N-1}} \mathsf{C}_N(s_N) \|_{A_N} \mathsf{C}_{N+1}(s_{N+1}) \xrightarrow{in(d)}_r$ $\mathsf{C}_0(s_0) \|_{A_0} \ldots \|_{A_{N-1}} \mathsf{C}_N(s_N) \|_{A_N} \mathsf{C}_{N+1}(d) \equiv I(s'd)$.

2. The case is similar to the previous one.

3. First, let $1 \leq j < N$, and let $I(s) \equiv I' \|_{A_{j-1}} I_j(s_j s_{j+1}) \|_{A_{j+1}} I''$, where
   $I' \equiv \mathsf{C}_0(s_0) \|_{A_0} \ldots \|_{A_{j-2}} \mathsf{C}_{j-1}(s_{j-1})$,
   $I_j(s_j s_{j+1}) \equiv \mathsf{C}_j(s_j) \|_{A_j} \mathsf{C}_{j+1}(s_{j+1})$,
   $I'' \equiv \mathsf{C}_{j+2}(s_{j+2}) \|_{A_{j+2}} \ldots \|_{A_N} \mathsf{C}_{N+1}(s_{N+1})$.
   If $s_j \in \{\bot, \underline{\bot}\}$ and $s_{j+1} \in \{d, \underline{d}\}$ then
   $I_j(s_j s_{j+1}) \equiv \mathsf{C}_j(s_j) \|_{A_j} \mathsf{C}_{j+1}(s_{j+1}) \xrightarrow{\delta_j(d)}_r \mathsf{C}_j(d) \|_{A_j} \mathsf{C}_{j+1}(\bot) \equiv I_j(d\ \bot)$ by 3.5.1 and 3.
   Since $\delta_j(d) \notin A_{j-1}$ and $\delta_j(d) \notin A_{j+1}$, we have $I(s) \xrightarrow{\delta_j(d)}_r I' \|_{A_{j-1}} I_j(d\ \bot) \|_{A_{j+1}} I'' \equiv I(s'd \perp s'')$.

   The other cases are similar; e.g. for $j = 0$, we have $I(s) \equiv \mathsf{C}_0(s_0) \|_{A_0} \mathsf{C}_1(s_1) \|_{A_1} I''$.
   If $s_0 \in \{\bot, \underline{\bot}\}$ and $s_1 \in \{d, \underline{d}\}$ then $\mathsf{C}_0(s_0) \|_{A_0} \mathsf{C}_1(s_1) \xrightarrow{\delta_0(d)}_r \mathsf{C}_0(d) \|_{A_0} \mathsf{C}_1(\bot)$. Since $\delta_0(d) \notin A_1$ $I(s) \xrightarrow{\delta_0(d)}_r \mathsf{C}_0(d) \|_{A_0} \mathsf{C}_1(\bot) \|_{A_1} I'' \equiv I(d \perp s'')$.

□

Finally, the transitional behaviour of Pipe can be given. It is a direct consequence of the previous proposition.

Proposition 3.7 Let $s = s_0 \ldots s_{N+1} \in D_\bot^{N+2}$ and $d \in$ V. Then, all action transitions of the process $\mathsf{Pipe}(s)$ are the following:

1. $s_{N+1} \in \{\bot, \underline{\bot}\}$ implies $\mathsf{Pipe}(s) \xrightarrow{in(d)}_r \mathsf{Pipe}(s'd)$ where $s' = s_0 \dots s_N$;

2. $s_0 \in \{d, \underline{d}\}$ implies $\mathsf{Pipe}(s) \xrightarrow{out(d)}_r \mathsf{Pipe}(\bot\ s')$, where $s' = s_1 \dots s_{N+1}$;

3. if there exists $j \in \{0, \dots, N\}$ such that $s_j \in \{\bot, \underline{\bot}\}$ and $s_{j+1} \in \{d, \underline{d}\}$ then $\mathsf{Pipe}(s) \xrightarrow{\tau}_r \mathsf{Pipe}(s'd \bot s'')$, where $s' = s_0 \dots s_{j-1}$ (which is ε for $j = 0$) and $s'' = s_{j+2} \dots s_{N+1}$ (which is ε for $j = N$).

We will apply Proposition 2.10 (in particular, part 4) to determine the time steps of $\mathsf{Pipe}(s)$; for this, the residual times of the components in Pipe are also needed, as we show first.

Proposition 3.8 Let $s = s_0 \dots s_{N+1} \in D_\bot^{N+2}$, $d \in$ V and $j \in \{0, \dots, N\}$. Then:

1. $\mathcal{R}(\delta_j(d), I(s)) = \mathcal{R}(\delta_j(d), \mathsf{C}_j(s_j) \|_{A_j} \mathsf{C}_{j+1}(s_{j+1}))$.

2. $\mathcal{R}(in(d), I(s)) = \mathcal{R}(in(d), \mathsf{C}_{N+1}(s_{N+1}))$.

3. $\mathcal{R}(out(d), I(s)) = \mathcal{R}(out(d), \mathsf{C}_0(s_0))$.

Proof:

1. Assume $1 \le j \le N-1$ and let $I(s) \equiv I' \|_{A_{j-1}} I_j(s_j s_{j+1}) \|_{A_{j+1}} I''$, where as above
   $I' \equiv \mathsf{C}_0(s_0) \|_{A_0} \dots \|_{A_{j-2}} \mathsf{C}_{j-1}(s_{j-1})$,
   $I_j(s_j s_{j+1}) \equiv \mathsf{C}_j(s_j) \|_{A_j} \mathsf{C}_{j+1}(s_{j+1})$,
   $I'' \equiv \mathsf{C}_{j+2}(s_{j+2}) \|_{A_{j+2}} \dots \|_{A_N} \mathsf{C}_{N+1}(s_{N+1}))$ and $s' = s_0 \dots s_{j-1}$, $s'' = s_{j+2} \dots s_{N+1}$. By Definition 2.4 and the fact that $\delta_j(d)$ is neither in A_{j-1} nor in A_{j+1} we have:
   $\mathcal{R}(\delta_j, I(s)) = \min\{\mathcal{R}(\delta_j(d), I' \|_{A_{j-1}} I_j(s_j s_{j+1})), \mathcal{R}(\delta_j(d), I'')\} =$
   $\min\{\min\{\mathcal{R}(\delta_j(d), I'), \mathcal{R}(\delta_j(d), I_j(s_j s_{j+1}))\}, \mathcal{R}(\delta_j(d), I'')\}\} =$
   $\min\{\mathcal{R}(\delta_j(d), I'), \mathcal{R}(\delta_j(d), I_j(s_j s_{j+1})), \mathcal{R}(\delta_j(d), I'')\}$.

   Consider I'. Since $\delta_j(d) \notin A_0, \dots, A_{j-2}$ and no $\mathsf{C}_k(s_k)$, $k \in \{0, \dots, j-2\}$, can perform $\mathsf{C}_k(s_k) \xrightarrow{\delta_j(d)}_r$ (so that also $I' \xrightarrow{\delta_j(d)}_r$ is not derivable) we have $\mathcal{R}(\delta_j(d), I') = 1$ (see Lemma 2.5.2). In a similar way one can prove that $\mathcal{R}(\delta_j(d), I'') = 1$. Hence, $\mathcal{R}(\delta_j(d), I(s)) = \min\{1, \mathcal{R}(\delta_j(d), I_j(s_j s_{j+1}), 1\} = \mathcal{R}(\delta_j(d), I_j(s_j s_{j+1}))$.

   Cases $j = 0$, $j = N$ can be proved similarly.

2. Since $in(d) \notin A_j$, $\mathcal{R}(in(d), I(s)) = \min\{\mathcal{R}(in(d), \mathsf{C}_j(s_j)) \mid j = 0, \dots, N+1\}$. Moreover, for $j = 0, \dots, N$, $in(d) \notin \mathcal{A}(\mathsf{C}_j(s_j))$. Hence $\mathcal{R}(in(d), \mathsf{C}_j(s_j)) = 1$ and $\mathcal{R}(in(d), I(s)) = \mathcal{R}(in(d), \mathsf{C}_{N+1}(s_{N+1}))$.

3. This case is similar to the previous one.

□

Proposition 3.9 Let $s = s_0 \dots s_{N+1} \in D_\bot^{N+2}$. Then:

1. $\mathcal{R}(\tau, \mathsf{Pipe}(s)) = 0$ iff $\exists j = 0, \dots, N$ such that $s_j = \underline{\bot}$ and $s_{j+1} = \underline{d}$ for some $d \in$ V;

2. for $d \in$ V, $\mathcal{R}(in(d), \mathsf{Pipe}(s)) = 0$ iff $s_{N+1} = \underline{\bot}$;

3. for $d \in$ V, $\mathcal{R}(out(d), \mathsf{Pipe}(s)) = 0$ iff $s_0 = \underline{d}$.

Proof:

1. $\mathcal{R}(\tau, \mathsf{Pipe}(s)) = \min\{\mathcal{R}(\beta, I(s)) \mid \beta \in \Phi_A^{-1}(\tau)\} = \min\{\mathcal{R}(\beta, I(s)) \mid \beta \in A\}$. Hence $\mathcal{R}(\tau, \mathsf{Pipe}(s)) = 0$ iff $\exists j = 0, \ldots, N$ such that $\mathcal{R}(\delta_j(d), I(s)) = 0$ for some $d \in$ V. (Observe that we can ignore $\beta = \tau$ by 2.5.2, since $I(s)$ cannot perform τ and thus $\mathcal{R}(\tau, I(s)) = 1$.) For fixed $d \in$ V, we have by Proposition 3.8,
$\mathcal{R}(\delta_j(d), I(s)) = \mathcal{R}(\delta_j(d), \mathsf{C}_j(s_j) \|_{A_j} \mathsf{C}_{j+1}(s_{j+1})) =$
$\max\{\mathcal{R}(\delta_j(d), \mathsf{C}_j(s_j)), \mathcal{R}(\delta_j(d), \mathsf{C}_{j+1}(s_{j+1}))\}$ since $\delta_j(d) \in A_j$. Thus,
$\mathcal{R}(\delta_j(d), I(s)) = 0$ iff $\mathcal{R}(\delta_j(d), \mathsf{C}_j(s_j)) = \mathcal{R}(\delta_j(d), \mathsf{C}_{j+1}(s_{j+1})) = 0$. Let us check when $\mathcal{R}(\delta_j(d), \mathsf{C}_j(s_j)) = 0$. $\mathcal{R}(\delta_j(d), \mathsf{C}_j(s_j)) = \min\{\mathcal{R}(\beta, \mathsf{C}(s_j)) \mid \beta \in \Phi_j^{-1}(\delta_j(d))\} = \mathcal{R}(in(d), \mathsf{C}(s_j))$. We distinguish three cases: $s_j \in D$, $s_j = \bot$ and $s_j = \underline{\bot}$. If $s_j \in D$, $in(d) \notin \mathcal{A}(\mathsf{C}(s_j))$ and $\mathcal{R}(in(d), \mathsf{C}(s_j)) = 1$. If $s_j = \bot$, Definition 3.3 and 2.4 (cases Pref and Sum) imply $\mathcal{R}(in(d), \mathsf{C}(s_j)) = 1$. Finally, if $s_j = \underline{\bot}$ then, again by Definition 2.4, we have $\mathcal{R}(in(d), \mathsf{C}(s_j)) = 0$. Then $\mathcal{R}(\delta_j(d), \mathsf{C}_j(s_j)) = 0$ iff $s_j = \underline{\bot}$.

   With similar reasoning we can prove that $\mathcal{R}(\delta_j(d), \mathsf{C}_{j+1}(s_{j+1})) = \mathcal{R}(out(d), \mathsf{C}(s_{j+1})) = 0$ iff $s_{j+1} = \underline{d}$ for $d \in$ V. We can conclude that $\mathcal{R}(\tau, \mathsf{Pipe}(s)) = 0$ iff $\exists j = 0, \ldots, N$ such that $s_j = \underline{\bot}$ and $s_{j+1} = \underline{d}$ for some $d \in$ V.

2. $\mathcal{R}(in(d), \mathsf{Pipe}(s)) = \min\{\mathcal{R}(\beta, I(s)) \mid \beta \in \Phi_A^{-1}(in(d))\} = \mathcal{R}(in(d), I(s))$. By Proposition 3.8, $\mathcal{R}(in(d), I(s)) = \mathcal{R}(in(d), \mathsf{C}_{N+1}(s_{N+1}))$. As in the previous item, we can prove that $\mathcal{R}(in(d), \mathsf{C}_{N+1}(s_{N+1})) = \min\{\mathcal{R}(\beta, \mathsf{C}(s_{N+1})) \mid \beta \in \Phi_{N+1}^{-1}(in(d))\} = \mathcal{R}(in(d), \mathsf{C}(s_{N+1}))$ and $\mathcal{R}(in(d), \mathsf{C}(s_{N+1})) = 0$ iff $s_{N+1} = \underline{\bot}$.

3. Similar to the previous case.

□

The above proposition provides us with the needed ingredients to show how $\mathsf{Pipe}(s)$ evolves by performing time steps, i.e. $\xrightarrow{X}_r$-transitions; recall, in particular, the definition of $\underline{s}$ for string s given at the beginning of this section.

Proposition 3.10 Let $s = s_0 \ldots s_{N+1} \in D_\bot^{N+2}$. If $s_j = \underline{\bot}$ and $s_{j+1} = \underline{d}$, for some $j = 0, \ldots, N$ and $d \in$ V, then $\mathsf{Pipe}(s) \not\xrightarrow{X}_r$ for every $X \subseteq \mathbb{A}$. Otherwise, the time steps of $\mathsf{Pipe}(s)$ are the following, where $a \in \text{V} \cup \{\bot\}$:

1. $s = \underline{a} s' \underline{\bot}$ implies $\mathsf{Pipe}(s) \xrightarrow{X}_r \mathsf{Pipe}(\underline{s})$, for every $X \subseteq \mathbb{A} \backslash \{out(a), in(0), in(1)\}$ (which is $\mathbb{A} \backslash \{in(0), in(1)\}$ for $a = \bot$);

2. $s = \underline{a} s' \bot$ implies $\mathsf{Pipe}(s) \xrightarrow{X}_r \mathsf{Pipe}(\underline{s})$, for every $X \subseteq \mathbb{A} \backslash \{out(a)\}$ (which is $\mathbb{A}$ for $a = \bot$);

3. $s = a s' \underline{\bot}$ implies $\mathsf{Pipe}(s) \xrightarrow{X}_r \mathsf{Pipe}(\underline{s})$, for every $X \subseteq \mathbb{A} \backslash \{in(0), in(1)\}$;

4. $s = a s' \bot$ implies $\mathsf{Pipe}(s) \xrightarrow{X}_r \mathsf{Pipe}(\underline{s})$, for every $X \subseteq \mathbb{A}$.

Proof: By Proposition 3.9.1, if $\exists j = 0, \ldots, N$ such that $s_j = \underline{\bot}$ and $s_{j+1} = \underline{d}$, then $\mathcal{R}(\tau, \mathsf{Pipe}(s)) = 0$. By Proposition 2.10.4 we have $\mathsf{Pipe}(s) \not\xrightarrow{X}_r$ for every $X \subseteq \mathbb{A}$.
Let us prove Item 1. By Definition 2.3, we have $\mathsf{Pipe}(\underline{a} s' \underline{\bot}) \overset{1}{\rightsquigarrow} \mathsf{Pipe}(\underline{a}\ \underline{s'}\ \underline{\bot})$. By 3.9 and the hypothesis (and 2.5.2), $\mathcal{R}(\alpha, \mathsf{Pipe}(s))$ is 1 for $\alpha \in \mathbb{A}_\tau \backslash \{out(a), in(0), in(1)\}$ and 0 otherwise. Then, always by Proposition 2.10.4, $\mathsf{Pipe}(s) \xrightarrow{X}_r \mathsf{Pipe}(\underline{s})$ iff $X \subseteq \mathbb{A} \backslash \{out(a), in(0), in(1)\}$.
The proof for the other items is completely analogous. □

3.3 Buffer "Buff"

Assume now that N cells are not connected end to end (as in Pipe) but are used as a storage. The cells interact with a centralized buffer controller which can store two more values. The software architecture of Buff is described in Fig. 3. We start by describing the functional behaviour of store Mem.

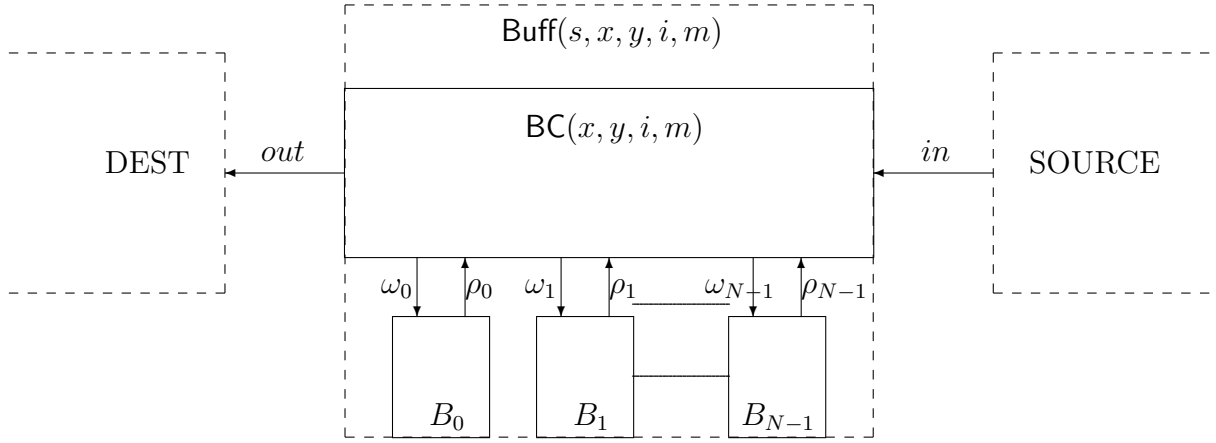

Figure 3: The Software Architecture for Buff

Definition 3.11 (*Mem*) Let $x \in D_\perp$ and $j = 0, \ldots, N-1$. The j-th element of Mem is described by process $B_j(x) \equiv \mathsf{C}(x)[\Phi'_j]$, where the various relabelling functions are defined as:

$$\Phi'_j(\alpha) = \begin{cases} \omega_j(0) & \text{if} \quad \alpha = in(0) \\ \omega_j(1) & \text{if} \quad \alpha = in(1) \\ \rho_j(0) & \text{if} \quad \alpha = out(0) \\ \rho_j(1) & \text{if} \quad \alpha = out(1) \\ \alpha & \text{otherwise} \end{cases}$$

Let $s = s_0 \ldots s_{N-1} \in D_\perp^N$. Define

$$\mathsf{Mem}(s) \equiv B_0(s_0) \|_\emptyset \ldots \|_\emptyset B_{N-1}(s_{N-1})$$

where again associativity holds.

Moreover, let $B = \{\omega_j(d), \rho_j(d) \mid d \in \mathrm{V}, j = 0, \ldots, N-1\}$.

The transitional properties of Mem immediately follow from its definition.

Proposition 3.12 Let $d \in \mathrm{V}$, $s = s_0 \ldots s_{N-1} \in D_\perp^N$ and $j = 0, \ldots, N-1$. All action transitions and relevant residual times are the following:

1. $s_j \in \{\perp, \underline{\perp}\}$ implies $\mathsf{Mem}(s) \xrightarrow{\omega_j(d)}_r \mathsf{Mem}(s')$ where $s' = s_0 \ldots s_{j-1} d s_{j+1} \ldots s_{N-1}$;
2. $s_j \in \{d, \underline{d}\}$ implies $\mathsf{Mem}(s) \xrightarrow{\rho_j(d)}_r \mathsf{Mem}(s')$, where $s' = s_0 \ldots s_{j-1} \perp s_{j+1} \ldots s_{N-1}$;
3. $\mathcal{R}(\omega_j(d), \mathsf{Mem}(s)) = \mathcal{R}(\omega_j(d), B_j(s_j))$, and $\mathcal{R}(\omega_j(d), B_j(s_j)) = 0$ iff $s_j = \underline{\perp}$;
4. $\mathcal{R}(\rho_j(d), \mathsf{Mem}(s)) = \mathcal{R}(\rho_j(d), B_j(s_j))$, and $\mathcal{R}(\rho_j(d), B_j(s_j)) = 0$ iff $s_j = \underline{d}$.

Proof:

1. Clearly, $\mathsf{Mem}(s)$ can only perform $\omega_j(d)$, if $B_j(s_j)$ can, i.e. if $s_j \in \{\bot, \underline{\bot}\}$.

   If $s_j \in \{\bot, \underline{\bot}\}$, then $B_j(s_j) \xrightarrow{\omega_j(d)}_r B_j(d)$ and, hence, $\mathsf{Mem}(s) \xrightarrow{\omega_j(d)}_r \mathsf{Mem}(s')$, where $\mathsf{Mem}(s') \equiv B_0(s_0) \parallel_\emptyset \ldots \parallel_\emptyset B_{j-1}(s_{j-1}) \parallel_\emptyset B_j(d) \parallel_\emptyset B_{j+1}(s_{j+1}) \ldots B_{N-1}(s_{N-1})$.

2. This case is similar to the previous one.

3. $\mathcal{R}(\omega_j(d), \mathsf{Mem}(s)) = \min\{\mathcal{R}(\omega_j(d), B_i(s_i)) \mid i = 0, \ldots, N-1\}$. If $i \neq j$ then $\omega_j(d) \notin \mathcal{A}(B_i(s_i))$ and $\mathcal{R}(\omega_j(d), B_i(s_i)) = 1$. Hence $\mathcal{R}(\omega_j(d), \mathsf{Mem}(s)) = \mathcal{R}(\omega_j(d), B_j(s_j))$. Moreover, $\mathcal{R}(\omega_j(d), B_j(s_j)) = \min\{\mathcal{R}(\alpha, \mathsf{C}(s_j)) \mid \alpha \in (\Phi'_j)^{-1}(\omega_j(d))\} = \mathcal{R}(in(d), \mathsf{C}(s_j)) = 0$ iff $s_j = \underline{\bot}$ (as in 3.9.1)

4. This case is similar to the previous one.

□

Mem acts as a store. It is used by a buffer controller (BC) to store data received from the external environment. The buffer controller uses the N cells of Mem as a circular queue (ordered as $0 < 1 < \ldots < N-1$). More in detail, the buffer controller accepts a value from the external environment and then writes the accepted value in the first available empty cell. It cannot accept any other value until the accepted one is actually stored in one of the N cells. The buffer controller also retains the oldest undelivered value and delivers it whenever possible.

The state of BC, $\mathsf{BC}(x, y, i, m)$ is determined by the four arguments:

(1) $x \in$ V, the value in input. That is, the value that BC has recently accepted from the environment; if $x = \bot$, then a new value can be accepted.

(2) $y \in$ V, the value in output. That is, the value read from Mem that can be made available to the external environment; if $y = \bot$, then no value in output is available.

(3) i, the index of the cell containing the oldest undelivered message.

(4) m, the number of messages in store Mem.

Since the messages are stored in contiguous cells of the circular queue, the first empty cell is given by $(i + m)\, mod\, N$, i.e. the sum of i and m modulo N. Finally, if n denotes the number of messages in the buffer, we always have $m \leq n \leq m + 2$.

BC can accept a new value from the environment only if $x = \bot$; on the contrary, if $x = d \in$ V, then BC can only write d into the first available cell of Mem (i.e. perform $\omega_{(i+m)\, mod\, N}(d)$). This storing operation is possible if $m < N$.

Similarly, BC can deliver a value only if $y = a \in$ V. Again, if $y = \bot$, then BC can only read the oldest undelivered value from Mem (i.e. perform $\rho_i(a)$). This operation is possible, if Mem is not empty, that is $m > 0$.

The buffer controller is then defined as follows.

Definition 3.13 (*Buffer Controller*) Let $d, a \in$ V, $0 \leq i \leq N-1$ and $0 \leq m \leq N$.

1. $\mathsf{BC}(\bot, \bot, i, 0) \equiv \sum_{d \in \mathrm{V}} in(d).\mathsf{BC}(d, \bot, i, 0)$;

2. $m > 0$ implies
   $\mathsf{BC}(\bot, \bot, i, m) \equiv (\sum_{d \in \mathrm{V}} in(d).\mathsf{BC}(d, \bot, i, m)) + (\sum_{a \in \mathrm{V}} \rho_i(a).\mathsf{BC}(\bot, a, (i+1)\, mod\, N, m-1))$;

3. $\mathsf{BC}(d, \bot, i, 0) \equiv \omega_i(d).\mathsf{BC}(\bot, \bot, i, 1)$;

4. $0 < m < N$ and $j = (i+m)\ mod N$ imply
   $\mathsf{BC}(d, \perp, i, m) \equiv \omega_j(d).\mathsf{BC}(\perp, \perp, i, m+1) + (\sum_{a \in \mathrm{V}} \rho_i(a).\mathsf{BC}(d, a, (i+1)\ mod N, m-1))$;
5. $\mathsf{BC}(d, \perp, i, N) \equiv \sum_{a \in \mathrm{V}} \rho_i(a).\mathsf{BC}(d, a, (i+1)\ mod N, N-1)$;
6. $\mathsf{BC}(\perp, a, i, m) \equiv (\sum_{d \in \mathrm{V}} in(d).\mathsf{BC}(d, a, i, m)) + out(a).\mathsf{BC}(\perp, \perp, i, m)$;
7. $m < N$ and $j = (i+m)\ mod N$ imply
   $\mathsf{BC}(d, a, i, m) \equiv \omega_j(d).\mathsf{BC}(\perp, a, i, m+1) + out(a).\mathsf{BC}(d, \perp, i, m)$;
8. $\mathsf{BC}(d, a, i, N) \equiv out(a).\mathsf{BC}(d, \perp, i, N)$.

Denote by $\underline{\mathsf{BC}}(x, y, i, m)$ the target process of a time-step out of $\mathsf{BC}(x, y, i, m)$. Hence:

1. $\underline{\mathsf{BC}}(\perp, \perp, i, 0) \equiv \sum_{d \in \mathrm{V}} \underline{in}(d).\mathsf{BC}(d, \perp, i, 0)$.
2. $0 < m$ implies
   $\underline{\mathsf{BC}}(\perp, \perp, i, m) \equiv (\sum_{d \in \mathrm{V}} \underline{in}(d).\mathsf{BC}(d, \perp, i, m)) + (\sum_{a \in \mathrm{V}} \underline{\rho_i}(a).\mathsf{BC}(\perp, a, (i+1)\ mod N, m-1))$.
3. $\underline{\mathsf{BC}}(d, \perp, i, 0) \equiv \underline{\omega_i}(d).\mathsf{BC}(\perp, \perp, i, 1)$.
4. $0 < m < N$ and $j = (i+m)\ mod N$ imply
   $\underline{\mathsf{BC}}(d, \perp, i, m) \equiv \underline{\omega_j}(d).\mathsf{BC}(\perp, \perp, i, m+1) + (\sum_{a \in \mathrm{V}} \underline{\rho_i}(a).\mathsf{BC}(d, a, (i+1)\ mod N, m-1))$.
5. $\underline{\mathsf{BC}}(d, \perp, i, N) \equiv \sum_{a \in \mathrm{V}} \underline{\rho_i}(a).\mathsf{BC}(d, a, (i+1)\ mod N, N-1)$.
6. $\underline{\mathsf{BC}}(\perp, a, i, m) \equiv (\sum_{d \in \mathrm{V}} \underline{in}(d).\mathsf{BC}(d, a, i, m)) + \underline{out}(a).\mathsf{BC}(\perp, \perp, i, m)$.
7. $m < N$ and $j = (i+m)\ mod N$ imply
   $\underline{\mathsf{BC}}(d, a, i, m) \equiv \underline{\omega_j}(d).\mathsf{BC}(\perp, a, i, m+1) + \underline{out}(a).\mathsf{BC}(d, \perp, i, m)$.
8. $\underline{\mathsf{BC}}(d, a, i, N) \equiv \underline{out}(a).\mathsf{BC}(d, \perp, i, N)$.

Process Buff is obtained by composing in parallel the memory Mem and the buffer controller BC. These two components synchronize on actions in B.

Definition 3.14 (Buff) We define $\mathsf{Buff} \equiv \mathsf{Buff}(\perp^N, \perp, \perp, 0, 0)$ based on the following: Let $x, y \in \mathrm{V} \cup \perp$, $s = s_0 \ldots s_{N-1} \in D_\perp^N$, $0 \le i \le N-1$ and $0 \le m \le N$ satisfy the *buff-invariant* that $s_j \in \{\perp, \underline{\perp}\}$ iff $j \notin \{i, i+1, \ldots, i+m-1\}$ (viewed $mod N$ and empty iff $m = 0$). Define

$$\mathsf{Buff}(s, x, y, i, m) \equiv (\mathsf{Mem}(s) \|_B \mathsf{BC}(x, y, i, m))/B$$

$$\underline{\mathsf{Buff}}(s, x, y, i, m) \equiv (\mathsf{Mem}(\underline{s}) \|_B \underline{\mathsf{BC}}(x, y, i, m))/B.$$

Observe that $\underline{\mathsf{Buff}}(\ldots)$ is a process where all components – in particular BC – have urgent actions to perform, independently of s. In $\mathsf{Buff}(\ldots)$, this is not the case for BC, but it might be the case for some components of Mem; this is indicated by underlined items in s. The following proposition states transitional properties of the Buff-processes.

Proposition 3.15 Let $a, d \in \mathrm{V}$ and s, i, m as in 3.14. All action transitions of $\mathsf{Buff}(s, d, a, i, m)$ are the following, where in particular the buff-invariant is preserved:

1. $\mathsf{Buff}(s, \perp, \perp, i, m) \xrightarrow{in(d)}_r \mathsf{Buff}(s, d, \perp, i, m)$;
2. $m > 0$ implies $\mathsf{Buff}(s, \perp, \perp, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', \perp, a, (i+1)\ mod N, m-1)$,
   if $s_i \in \{a, \underline{a}\}$ and $s' = s_0 \ldots s_{i-1} \perp s_{i+1} \ldots s_{N-1}$;

3. $m > 0$ implies $\mathsf{Buff}(s, d, \bot, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', d, a, (i+1)\, mod N, m-1)$, if $s_i \in \{a, \underline{a}\}$ and $s' = s_0 \dots s_{i-1} \bot s_{i+1} \dots s_{N-1}$;

4. $m < N$ implies $\mathsf{Buff}(s, d, \bot, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, \bot, i, m+1)$, if $s' = s_0 \dots s_{j-1} d s_{j+1} \dots s_{N-1}$ and $j = (i+m)\, mod N$;

5. $\mathsf{Buff}(s, \bot, a, i, m) \xrightarrow{in(d)}_r \mathsf{Buff}(s, d, a, i, m)$;

6. $\mathsf{Buff}(s, \bot, a, i, m) \xrightarrow{out(a)}_r \mathsf{Buff}(s, \bot, \bot, i, m)$;

7. $\mathsf{Buff}(s, d, a, i, m) \xrightarrow{out(a)}_r \mathsf{Buff}(s, d, \bot, i, m)$;

8. $m < N$ implies $\mathsf{Buff}(s, d, a, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, a, i, m+1)$, if $s' = s_0 \dots s_{j-1} d s_{j+1} \dots s_{N-1}$ and $j = (i+m)\, mod N$.

Proof: To see that no other action transitions are possible, observe that all actions of $\mathsf{Mem}(s)$ are synchronized with BC (and hidden afterwards), hence an action can only be performed if BC takes part in it.

1. By Definition 3.13, we have $\mathsf{BC}(\bot, \bot, i, m) \xrightarrow{in(d)}_r \mathsf{BC}(d, \bot, i, m)$ for all $d \in \mathrm{V}$. $in(d) \notin B$ implies $\mathsf{Mem}(s) \parallel_B \mathsf{BC}(\bot, \bot, i, m) \xrightarrow{in(d)}_r \mathsf{Mem}(s) \parallel_B \mathsf{BC}(d, \bot, i, m)$ and $\mathsf{Buff}(s, \bot, \bot, i, m) \xrightarrow{\Phi_B(in(d))}_r \mathsf{Buff}(s, d, \bot, i, m)$. With $\Phi_B(in(d)) = in(d)$, this concludes the proof.

2. The only other action transitions of $\mathsf{BC}(\bot, \bot, i, m)$ are $\mathsf{BC}(\bot, \bot, i, m) \xrightarrow{\rho_i(a)}_r \mathsf{BC}(\bot, a, (i+1)\, mod N, m-1)$ in case that $m > 0$. Moreover, by Proposition 3.12.2, $\mathsf{Mem}(s) \xrightarrow{\rho_i(a)}_r \mathsf{Mem}(s')$ for some $a \in \mathrm{V}$ iff $s_i = a$ and $s' = s_0 \dots s_{i-1} \bot s_{i+1} \dots s_{N-1}$. Hence $\mathsf{Mem}(s) \parallel_B \mathsf{BC}(\bot, \bot, i, m) \xrightarrow{\rho_i(a)}_r \mathsf{Mem}(s') \parallel_B \mathsf{BC}(\bot, a, (i+1)\, mod N, m-1)$ and $\mathsf{Buff}(s, \bot, \bot, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, a, (i+1)\, mod N, m-1)$.

3. This case is similar to the previous one.

4. $\mathsf{BC}(d, \bot, i, m)$ can only perform the actions $\rho_i(a)$, $a \in \mathrm{V}$, in case $m > 0$, which are treated in Part 3, or the action $\omega_j(d)$ in case $m < N$, which we treat now. We have $\mathsf{BC}(d, \bot, i, m) \xrightarrow{\omega_j(d)}_r \mathsf{BC}(\bot, \bot, i, m+1)$. Moreover, by the Proposition 3.12.1 and the buff-invariant, $\mathsf{Mem}(s) \xrightarrow{\omega_j(d)}_r \mathsf{Mem}(s')$ where $s' = s_0 \dots s_{j-1} d s_{j+1} \dots s_{N-1}$. Then $\mathsf{Mem}(s) \parallel_B \mathsf{BC}(d, \bot, i, m) \xrightarrow{\omega_j(d)}_r \mathsf{Mem}(s') \parallel_B \mathsf{BC}(\bot, \bot, i, m+1)$ and $\mathsf{Buff}(s, d, \bot, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, \bot, i, m+1)$.

All the other cases can be proved similarly. □

The following proposition deals with $\underline{\mathsf{Buff}}$ in place of Buff. We omit the proof, since it is completely analogous to the proof of Proposition 3.15.

Proposition 3.16 Let $a, d \in \mathrm{V}$ and s, i, m as in 3.14. All action transitions of $\underline{\mathsf{Buff}}(s, d, a, i, m)$ are the following, where in particular the buff-invariant is preserved:

1. $\underline{\mathsf{Buff}}(s, \bot, \bot, i, m) \xrightarrow{in(d)}_r \mathsf{Buff}(\underline{s}, d, \bot, i, m)$;

2. $m > 0$ implies $\underline{\mathsf{Buff}}(s, \bot, \bot, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, a, (i+1)\, mod N, m-1)$, if $s_i \in \{a, \underline{a}\}$ and $s' = \underline{s_0} \dots \underline{s_{i-1}} \bot \underline{s_{i+1}} \dots \underline{s_{N-1}}$;

3. $m > 0$ implies $\underline{\mathsf{Buff}}(s, d, \bot, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', d, a, (i+1)\, mod N, m-1)$, if $s_i \in \{a, \underline{a}\}$ and $s' = \underline{s_0} \ldots \underline{s_{i-1}} \bot \underline{s_{i+1}} \ldots \underline{s_{N-1}}$;

4. $m < N$ implies $\underline{\mathsf{Buff}}(s, d, \bot, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, \bot, i, m+1)$, if $s' = \underline{s_0} \ldots \underline{s_{j-1}} d \underline{s_{j+1}} \ldots \underline{s_{N-1}}$ and $j = (i+m)\, mod N$;

5. $\underline{\mathsf{Buff}}(s, \bot, a, i, m) \xrightarrow{in(d)}_r \mathsf{Buff}(\underline{s}, d, a, i, m)$;

6. $\underline{\mathsf{Buff}}(s, \bot, a, i, m) \xrightarrow{out(a)}_r \mathsf{Buff}(\underline{s}, \bot, \bot, i, m)$;

7. $\underline{\mathsf{Buff}}(s, d, a, i, m) \xrightarrow{out(a)}_r \mathsf{Buff}(\underline{s}, d, \bot, i, m)$;

8. $m < N$ implies $\underline{\mathsf{Buff}}(s, d, a, i, m) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, a, i, m+1)$, if $s' = \underline{s_0} \ldots \underline{s_{j-1}} d \underline{s_{j+1}} \ldots \underline{s_{N-1}}$ and $j = (i+m)\, mod N$.

Again, we determine the residual times as needed.

Proposition 3.17 Let $a, d \in \mathrm{V}$ and x, y, s, i, m as in 3.14.

1. $\mathcal{R}(in(d), \mathsf{Buff}(s, x, y, i, m)) = 1$;
2. $\mathcal{R}(out(a), \mathsf{Buff}(s, x, y, i, m)) = 1$;
3. $\mathcal{R}(\tau, \mathsf{Buff}(s, x, y, i, m)) = 1$;
4. $\mathcal{R}(in(d), \underline{\mathsf{Buff}}(s, x, y, i, m)) = 0$ iff $x = \bot$;
5. $\mathcal{R}(out(a), \underline{\mathsf{Buff}}(s, x, y, i, m)) = 0$ iff $y = a$;
6. $\mathcal{R}(\tau, \underline{\mathsf{Buff}}(s, x, y, i, m)) = 0$ iff ($m > 0$ and $y = \bot$) or ($m < N$ and $x \in \mathrm{V}$).

Proof:

1. By definition of $\mathcal{R}(\alpha, P)$, in particular the Rel case, we have
$\mathcal{R}(in(d), \mathsf{Buff}(s, x, y, i, m)) =$
$\min\{\mathcal{R}(\beta, \mathsf{Mem}(s) \|_B \mathsf{BC}(x, y, i, m)) \mid \beta \in \Phi_B^{-1}(in(d))\} =$
$\mathcal{R}(in(d), \mathsf{Mem}(s) \|_B \mathsf{BC}(x, y, i, m)) =$
$\min\{\mathcal{R}(in(d), \mathsf{Mem}(s)), \mathcal{R}(in(d), \mathsf{BC}(x, y, i, m))\}$
since $\Phi_B^{-1}(in(d)) = \{in(d)\}$ and $in(d) \notin B$. Moreover, $in(d) \notin \mathcal{A}(\mathsf{Mem}(s))$ and hence $\mathcal{R}(in(d), \mathsf{Mem}(s)) = 1$. Then $\mathcal{R}(in(d), \mathsf{Buff}(s, x, y, i, m)) =$
$\min\{1, \mathcal{R}(in(d), \mathsf{BC}(x, y, i, m))\} = \mathcal{R}(in(d), \mathsf{BC}(x, y, i, m))$. By the fact that BC is an initial process, we can conclude that $\mathcal{R}(in(d), \mathsf{Buff}(s, x, y, i, m)) = 1$.

2. This case is similar to the previous one.

3. Let $P \equiv \mathsf{Mem}(s) \|_B \mathsf{BC}(x, y, i, m)$ and $j = (i+m)\, mod N$. $\mathcal{R}(\tau, \mathsf{Buff}(s, x, y, i, m)) =$
$\min\{\mathcal{R}(\beta, P) \mid \beta \in \Phi_B^{-1}(\tau)\} = \min\{\mathcal{R}(\beta, P) \mid \beta \in B\}$. If $k \neq j$, then $P \not\xrightarrow{\omega_k(d)}_r$ and $\mathcal{R}(\omega_k(d), P) = 1$ for all $d \in \mathrm{V}$. Analogously, the residual time is 1 for all $\omega_j(d)$ with $d \neq x$ and all $\rho_k(a)$ with $k \neq i$ or $s_i \notin \{a, \underline{a}\}$. It remains to show that the residual time is also 1 for the possibly remaining cases: If $x \in \mathrm{V}$, then we have $\mathcal{R}(\omega_j(x), P) = \max\{\mathcal{R}(\omega_j(x), \mathsf{Mem}(s)), \mathcal{R}(\omega_j(x), \mathsf{BC}(x, y, i, m))\} = 1$, since $\omega_j(x) \in B$ and BC is an initial process; the treatment of $\rho_i(a)$ is similar. We can conclude that $\mathcal{R}(\tau, \mathsf{Buff}(s, x, y, i, m)) = 1$.

4. Similarly to Item 1., we have $\mathcal{R}(in(d), \underline{\mathsf{Buff}}(s,x,y,i,m)) = \mathcal{R}(in(d), \underline{\mathsf{BC}}(x,y,i,m))$. It is left to prove that $\mathcal{R}(in(d), \underline{\mathsf{BC}}(x,y,i,m)) = 0$ iff $x = \perp$. Assume that $x = d' \in \mathrm{V}$. By the definition of the buffer controller, we have $\underline{\mathsf{BC}}(d',y,i,m) \stackrel{in(d)}{\not\longrightarrow}_r$ and hence $\mathcal{R}(in(d), \underline{\mathsf{BC}}(d',y,i,m)) = 1$; on the other hand, by the definition of $\mathcal{R}$, we get $\mathcal{R}(in(d), \underline{\mathsf{BC}}(\perp,y,i,m)) = 0$.

5. This case is similar to the previous one.

6. Let $P \equiv \mathsf{Mem}(\underline{s}) \parallel_B \underline{\mathsf{BC}}(x,y,i,m)$ and $j = (i+m)\, mod N$. As in the third item, we have that $\mathcal{R}(\tau, \underline{\mathsf{Buff}}(s,x,y,i,m))$ is the minimum of various residual times that are all 1, except possibly for $\mathcal{R}(\omega_j(x), P)$ if $x \in \mathrm{V}$ and $\mathcal{R}(\rho_i(a), P)\}$ if $a \in \mathrm{V}$ and $s_i \in \{a, \underline{a}\}$. Hence, $\mathcal{R}(\tau, \underline{\mathsf{Buff}}(s,x,y,i,m)) = 0$ iff one of these times is 0; we only treat the first case, since the second is similar, and may therefore assume $x \in \mathrm{V}$. Since $\omega_j(x) \in B$, $\mathcal{R}(\omega_j(x), P) = \max\{\mathcal{R}(\omega_j(x), \mathsf{Mem}(\underline{s})), \mathcal{R}(\omega_j(x), \underline{\mathsf{BC}}(x,y,i,m))\} = 0$ iff $\mathcal{R}(\omega_j(x), \mathsf{Mem}(\underline{s})) = \mathcal{R}(\omega_j(x), \underline{\mathsf{BC}}(x,y,i,m)) = 0$. By Proposition 3.12, $\mathcal{R}(\omega_j(d), \mathsf{Mem}(\underline{s})) = 0$ iff $\underline{s_j} = \underline{\perp}$, which is true by the buff-invariant. Moreover, by definition of the buffer controller we have that $m = N$ (or $x = \perp$) implies $\underline{\mathsf{BC}}(x,y,i,m) \stackrel{\omega_j(x)}{\not\longrightarrow}_r$ and hence $\mathcal{R}(\omega_j(x), \underline{\mathsf{BC}}(x,y,i,m)) = 1$; on the contrary, if $m < N$, $\mathcal{R}(\omega_j(x), \underline{\mathsf{BC}}(x,y,i,m)) = 0$. We can conclude that $\mathcal{R}(\omega_j(x), P) = 0$ iff $m < N$.

□

We are now ready to state how the Buff-processes evolve by performing $\stackrel{X}{\longrightarrow}_r$-transitions.

Proposition 3.18 Let $a, d \in \mathrm{V}$ and x, y, s, i, m as in 3.14. All the time-steps of the Buff-processes are the following.

1. $\mathsf{Buff}(s,x,y,i,m) \stackrel{X}{\longrightarrow}_r \underline{\mathsf{Buff}}(s,x,y,i,m)$ for every $X \subseteq \mathbb{A}$.
2. $m > 0$ implies $\underline{\mathsf{Buff}}(s,\perp,\perp,i,m) \stackrel{X}{\not\longrightarrow}_r$ for every $X \subseteq \mathbb{A}$.
3. $\underline{\mathsf{Buff}}(s,\perp,\perp,i,0) \stackrel{X}{\longrightarrow}_r \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)$ for every $X \subseteq \mathbb{A}\backslash\{in(0), in(1)\}$.
4. $\underline{\mathsf{Buff}}(s,\perp,a,i,m) \stackrel{X}{\longrightarrow}_r \underline{\mathsf{Buff}}(s,\perp,a,i,m)$ for every $X \subseteq \mathbb{A}\backslash\{out(a), in(0), in(1)\}$.
5. $\underline{\mathsf{Buff}}(s,d,\perp,i,m) \stackrel{X}{\not\longrightarrow}_r$ for every $X \subseteq \mathbb{A}$.
6. $0 \leq m < N$ implies $\underline{\mathsf{Buff}}(s,d,a,i,m) \stackrel{X}{\not\longrightarrow}_r$ for every $X \subseteq \mathbb{A}$.
7. $\underline{\mathsf{Buff}}(s,d,a,i,N) \stackrel{X}{\longrightarrow}_r \underline{\mathsf{Buff}}(s,d,a,i,N)$ for every $X \subseteq \mathbb{A}\backslash\{out(a)\}$.

Proof: We only prove the more involved cases. The other ones are similar.

1. By the definitions of $\stackrel{1}{\rightsquigarrow}$ and Buff and its components, $\mathsf{Buff}(s,x,y,i,m) \stackrel{1}{\rightsquigarrow} \underline{\mathsf{Buff}}(s,x,y,i,m)$. Moreover, by 3.17.1, .2 and .3, $\mathcal{R}(\alpha, \mathsf{Buff}(s,x,y,i,m)) = 1$ for every $\alpha \in \mathbb{A}_\tau$. Then, by Proposition 2.10, $\mathsf{Buff}(s,x,y,i,m) \stackrel{X}{\longrightarrow}_r \underline{\mathsf{Buff}}(s,x,y,i,m)$ for every $X \subseteq \mathbb{A}$.
2. If $m > 0$, then by 3.17.6 we have $\mathcal{R}(\tau, \underline{\mathsf{Buff}}(s,\perp,\perp,i,m)) = 0$ and hence, by Proposition 2.10, $\underline{\mathsf{Buff}}(s,\perp,\perp,i,m) \stackrel{X}{\not\longrightarrow}_r$ for every $X \subseteq \mathbb{A}$.
3. $\underline{\mathsf{Buff}}(s,\perp,\perp,i,0) \stackrel{1}{\rightsquigarrow} \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)$. Moreover, $\mathcal{R}(\alpha, \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)) = 1$ by 3.17.5 and .6 for any $\alpha \notin \{in(0), in(1)\}$. Again by Proposition 2.10, $\underline{\mathsf{Buff}}(s,\perp,\perp,i,0) \stackrel{X}{\longrightarrow}_r \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)$ for every $X \subseteq \mathbb{A}\backslash\{in(0), in(1)\}$.
5. By 3.17.6 we have $\mathcal{R}(\tau, \underline{\mathsf{Buff}}(s,d,\perp,i,m)) = 0$ since $m > 0$ of $m < N$, and conclude that $\underline{\mathsf{Buff}}(s,d,\perp,i,m) \stackrel{X}{\not\longrightarrow}_r$ for every $X \subseteq \mathbb{A}$.

□

4 Comparing the three buffers

This section is the core of the paper. We compare the three buffer implementations with respect to the efficiency preorder introduced in Section 2.1. To do this, we will exploit the alternative characterization in terms of refusal traces.

Given two processes P and Q, to prove that P is not more efficient than Q, $P \not\sqsupseteq Q$, we exhibit a refusal trace of P that Q cannot perform. This is sufficient since, by Proposition 2.11, $P \not\leq_r Q$ (that is $\mathsf{RT}(P) \not\subseteq \mathsf{RT}(Q)$) implies $P \not\sqsupseteq Q$. On the other hand, to prove that $P \sqsupseteq Q$, we show again by Proposition 2.11, that $P \leq_r Q$, i.e. $\mathsf{RT}(P) \subseteq \mathsf{RT}(Q)$. To prove this turns out to be a little bit more involved. It is well-known, however, that trace inclusion can be shown by exhibiting a suitable simulation relation. In our setting, to prove $\mathsf{RT}(P) \subseteq \mathsf{RT}(Q)$, we give a simulation relation between states of the refusal transitional semantics of P and of Q. A simulation relation is defined as follows.

A relation $\mathcal{R}$ is a *simulation relation* for two processes P and Q if $(P, Q) \in \mathcal{R}$ and whenever $(R, S) \in \mathcal{R}$ and $R \xrightarrow{\mu}_r R'$, where either $\mu = \alpha \in \mathbb{A}_\tau$ or $\mu = X \subseteq \mathbb{A}$, then either $S \stackrel{\mu}{\Rightarrow}_r S'$ if $\mu \neq \tau$ or $S \stackrel{\varepsilon}{\Rightarrow}_r S'$ if $\mu = \tau$ and in either case $(R', S') \in \mathcal{R}$.

The existence of a simulation relation relating two processes P and Q ensures $\mathsf{RT}(P) \subseteq \mathsf{RT}(Q)$ and hence $P \leq_r Q$.

4.1 Relating Fifo and Pipe

We start with $\mathsf{Fifo} \equiv \mathsf{Fifo}(\varepsilon)$ and $\mathsf{Pipe} \equiv \mathsf{Pipe}(\bot^{N+2})$ (for our fixed positive N) and prove that they are unrelated.

Theorem 4.1 1. $\mathsf{Pipe} \not\sqsupseteq \mathsf{Fifo}$, and

2. $\mathsf{Fifo} \not\sqsupseteq \mathsf{Pipe}$.

Proof: We have to prove that

1. $\mathsf{RT}(\mathsf{Pipe}(\bot^{N+2})) \not\subseteq \mathsf{RT}(\mathsf{Fifo}(\varepsilon))$, and
2. $\mathsf{RT}(\mathsf{Fifo}(\varepsilon)) \not\subseteq \mathsf{RT}(\mathsf{Pipe}(\bot^{N+2}))$.

To prove Item 1., we consider the refusal trace $v_1 = in(0)\emptyset\{out(0)\}$.

Then $v_1 \in \mathsf{RT}(\mathsf{Pipe}(\bot^{N+2}))$ since (see Proposition 3.7 and 3.10) $\mathsf{Pipe}(\bot^{N+2}) \xrightarrow{in(0)}_r \mathsf{Pipe}(\bot^{N+1}\ 0) \xrightarrow{\emptyset}_r \mathsf{Pipe}(\underline{\bot}^{N+1}\underline{0}) \xrightarrow{\tau}_r \mathsf{Pipe}(\underline{\bot}^{N}\ 0\ \bot) \xrightarrow{\{out(0)\}}_r$. On the other hand, $\mathsf{Fifo}(\varepsilon) \xrightarrow{in(0)}_r \mathsf{Fifo}(0) \xrightarrow{\emptyset}_r \underline{\mathsf{Fifo}}(0) \not\xrightarrow{\{out(0)\}}_r$ by Proposition 3.2.

Hence $\mathsf{RT}(\mathsf{Pipe}(\bot^{N+2})) \not\subseteq \mathsf{RT}(\mathsf{Fifo}(\varepsilon))$.

To prove Item 2., we consider the refusal trace $v_2 = in(0)\emptyset\emptyset out(0)\{in(0)\}$. Then $v_2 \in \mathsf{RT}(\mathsf{Fifo}(\varepsilon))$ since $\mathsf{Fifo}(\varepsilon) \xrightarrow{in(0)}_r \mathsf{Fifo}(0) \xrightarrow{\emptyset}_r \underline{\mathsf{Fifo}}(0) \xrightarrow{\emptyset}_r \underline{\mathsf{Fifo}}(0) \xrightarrow{out(0)}_r \mathsf{Fifo}(\varepsilon) \xrightarrow{\{in(0)\}}_r$ by Proposition 3.2.

We apply again Proposition 3.7 and 3.10 to see that this refusal trace cannot be performed by Pipe. The proof is more involved because the internal actions lead to a number of different cases. In order to mimic the execution of v_2, process Pipe must perform: $\mathsf{Pipe}(\bot^{N+2}) \xrightarrow{in(0)} \mathsf{Pipe}(\bot^{N+1}\ 0)$.

Now $\mathsf{Pipe}(\bot^{N+1}\ 0)$ can perform some internal actions before the time step, i.e. we have $\mathsf{Pipe}(\bot^{N+1}\ 0) \xrightarrow{w_1}_r \mathsf{Pipe}(\bot^i\ 0\ \bot^j)$, where $w_1 \in \{\tau\}^*$ and $i + j = N + 1$, and then $\mathsf{Pipe}(\bot^i\ 0\ \bot^j) \xrightarrow{\emptyset}_r \mathsf{Pipe}(\underline{\bot}^i\underline{0}\ \underline{\bot}^j)$. First, assume $i > 0$. By the first part of 3.10, $\mathsf{Pipe}(\underline{\bot}^i\underline{0}\ \underline{\bot}^j)$ has to perform at least one (urgent) τ before the next time step, i.e. $\mathsf{Pipe}(\underline{\bot}^i\underline{0}\ \underline{\bot}^j) \xrightarrow{w_2}_r \mathsf{Pipe}(\underline{\bot}^k 0\ \bot^{i-k}\ \ \underline{\bot}^j) \xrightarrow{\emptyset}_r \mathsf{Pipe}(\underline{\bot}^k\ \underline{0}\ \underline{\bot}^{j+i-k})$, where $w_2 = \tau^{i-k}$ and $j + i - k > 0$.

Now, all possible internal actions have to be performed such that 0 can be delivered; i.e. for $w_3 = \tau^k$, $\mathsf{Pipe}(\underline{\perp}^k\ \underline{0}\ \underline{\perp}^{j+i-k}) \xrightarrow{w_3}_r \mathsf{Pipe}(0\ \perp^k\ \underline{\perp}^{j+i-k}) \xrightarrow{out(0)}_r \mathsf{Pipe}(\perp^{k+1}\ \underline{\perp}^{j+i-k})$. By Proposition 3.10.3, $\mathsf{Pipe}(\perp^{k+1}\ \underline{\perp}^{j+i-k}) \not\xrightarrow{\{in(0)\}}_r$.

Finally, for $i = 0$, the only possible run fails as follows: $\mathsf{Pipe}(\perp^{N+1}\ 0) \xrightarrow{w_1}_r \mathsf{Pipe}(0\ \perp^{N+1}) \xrightarrow{\emptyset}_r \mathsf{Pipe}(\underline{0}\ \underline{\perp}^{N+1}) \xrightarrow{\emptyset}_r \mathsf{Pipe}(\underline{0}\ \underline{\perp}^{N+1}) \xrightarrow{out(0)}_r \mathsf{Pipe}(\perp\ \underline{\perp}^{N+1}) \not\xrightarrow{\{in(0)\}}_r$.

□

Theorem 4.1 shows that the worst-case behaviours of Fifo and Pipe are unrelated. The intuition behind these results is:

- Besides input and output of values, process Pipe must perform internal activities in order to manage the queue of cells (i.e. to move values from a cell to the next one). If Pipe receives a value d, then this value must pass through the $N + 2$ cells before being delivered, that is before an $out(d)$-action becomes available, and each move might take time 1. This is also the case, if d is the only value in Pipe. In this situation, Fifo would be ready to deliver d right after d is received and time 1 has passed.

- Conversely, after the execution of an *in*- or *out*-action, Fifo reaches an initial state, a state in which it can let time pass in any context. This is not necessarily true for Pipe: for example, $\mathsf{Pipe}(\underline{0\perp}^{N+1})$ can perform $out(0)$ and reach a state $P' \equiv \mathsf{Pipe}(\perp\ \underline{\perp}^{N+1})$ in which $\mathcal{R}(in(d), P') = 0$. In state P', buffer Pipe cannot let time pass if the environment offers some input while Fifo, in the corresponding state $\underline{\mathsf{Fifo}}(\varepsilon)$, can make a time step under any circumstance.

  More intuitively speaking, Pipe is a distributed implementation, where in particular input and output are independent activities; in Fifo, these are sequential, they block each other. This point, and its effect on the efficiency, might be easily overlooked when comparing Fifo and Pipe without a formal treatment.

4.2 Relating Fifo and Buff

The following theorem relates Fifo and $\mathsf{Buff} \equiv \mathsf{Buff}(\perp^N, \perp, \perp, 0, 0)$ for our fixed positive N.

Theorem 4.2 1. $\mathsf{Buff} \not\sqsupseteq \mathsf{Fifo}$

2. $\mathsf{Fifo} \sqsupseteq \mathsf{Buff}$

Proof: The proof of Item 1 is very similar to the previous proof. To prove that $\mathsf{RT}(\mathsf{Buff}(\perp^N, \perp, \perp, 0, 0)) \not\subseteq \mathsf{RT}(\mathsf{Fifo}(\varepsilon))$, we consider again $v = in(0)\emptyset\{out(0)\}$. This is a refusal trace of Buff, due to $\mathsf{Buff}(\perp^N, \perp, \perp, 0, 0) \xrightarrow{in(0)}_r \mathsf{Buff}(\perp^N, 0, \perp, 0, 0) \xrightarrow{\emptyset}_r \underline{\mathsf{Buff}}(\perp^N, 0, \perp, 0, 0) \xrightarrow{\tau}_r \mathsf{Buff}(0\underline{\perp}^{N-1}, \perp, \perp, 0, 1) \xrightarrow{\{out(0)\}}_r$ (see 3.15, 3.16 and 3.18). On the other hand, $v \notin \mathsf{RT}(\mathsf{Fifo}(\varepsilon))$ as shown above.

Let us concentrate now on Item 2. To prove that $\mathsf{RT}(\mathsf{Fifo}(\varepsilon)) \subseteq \mathsf{RT}(\mathsf{Buff}(\perp^N, \perp, \perp, 0, 0))$, we present a simulation relation for $\mathsf{Fifo}(\varepsilon)$ and $\mathsf{Buff}(\perp^N, \perp, \perp, 0, 0)$.

To this aim we need some further notation. Given the string s of Buff, the index i of the oldest undelivered value and the number of undelivered values, we define a function g that gives the sequence, in the order of their arrivals, of undelivered messages contained in Buff (g abstracts away from the location of the stored messages). A similar function was defined in [1]. Function g is used to classify significant states of Buff when looking for a simulation relation for Pipe and Buff.

Function $g : D_\perp^N \times \{0,\ldots,N-1\} \times \{0,\ldots,N\} \to \mathrm{V}_N$ is defined for $s =\in D_\perp^N$ (which we always take to be $s = s_0 \ldots s_{N-1}$), $i \in \{0,\ldots,N-1\}$ and $m \in \{0,\ldots,N\}$ by:

$$g(s,i,m) = \begin{cases} \varepsilon & \text{if } \ m = 0 \\ x_i & \text{if } \ m = 1 \\ x_i\, x_{(i+1)\, mod N} \cdots x_{(i+m-1)\, mod N} & \text{otherwise} \end{cases}$$

where $x_j = d$ if $s_j \in \{d, \underline{d}\}$ and $j = i, \ldots, (i+m-1)\, mod N$.

Our simulation relation is $\Re = \mathcal{A} \cup \underline{\mathcal{A}} \cup \mathcal{B} \cup \underline{\mathcal{B}} \cup \mathcal{C} \cup \underline{\mathcal{C}}$, where
$\mathcal{A} = \{(\mathsf{Fifo}(\varepsilon), \mathsf{Buff}(s,\perp,\perp,i,0)) \mid i \in [0..N] \text{ and } s \in \{\perp, \underline{\perp}\}^N$;
$\underline{\mathcal{A}} = \{(\underline{\mathsf{Fifo}}(\varepsilon), \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)) \mid i \in [0..N] \text{ and } s \in \{\perp, \underline{\perp}\}^N\}$;
$\mathcal{B} = \{(\mathsf{Fifo}(at), \mathsf{Buff}(s,\perp,a,i,|t|)) \mid i \in [0..N],\ s \in D_\perp^N,\ t \in \mathrm{V}_N,\ \text{and } g(s,i,|t|) = t\}$;
$\underline{\mathcal{B}} = \{(\underline{\mathsf{Fifo}}(at), \underline{\mathsf{Buff}}(s,\perp,a,i,|t|)) \mid i \in [0..N],\ s \in D_\perp^N,\ t \in \mathrm{V}_N,\ \text{and } g(s,i,|t|) = t\}$;
$\mathcal{C} = \{(\mathsf{Fifo}(atd), \mathsf{Buff}(s,d,a,i,N)) \mid i \in [0..N],\ s \in D_\perp^N,\ t \in \mathrm{V}_N,\ |t| = N \text{ and } g(s,i,N) = t\}$;
$\underline{\mathcal{C}} = \{(\underline{\mathsf{Fifo}}(atd), \underline{\mathsf{Buff}}(s,d,a,i,N)) \mid i \in [0..N],\ s \in D_\perp^N,\ t \in \mathrm{V}_N,\ |t| = N \text{ and } g(s,i,N) = t\}$.

Obviously, $(\mathsf{Fifo}(\varepsilon), \mathsf{Buff}(s,\perp,\perp,0,0)) \in \Re$ and we prove that $\Re$ is a simulation relation.

Consider a generic pair $(\mathsf{Fifo}(\varepsilon), \mathsf{Buff}(s,\perp,\perp,i,0))$ in $\mathcal{A}$ (hence in $\Re$) and the transitions that $\mathsf{Fifo}(\varepsilon)$ can perform (see Proposition 3.2). Assume:

(i) $\mathsf{Fifo}(\varepsilon) \xrightarrow{in(d)}_r \mathsf{Fifo}(d)$. By the buff-invariant (see Definition 3.14), $s \in \{\perp, \underline{\perp}\}^N$; consider the following transitions $\mathsf{Buff}(s,\perp,\perp,i,0) \xrightarrow{in(d)}_r \mathsf{Buff}(s,d,\perp,i,0) \xrightarrow{\tau}_r \mathsf{Buff}(s',\perp,\perp,i,1) \xrightarrow{\tau}_r \mathsf{Buff}(s'',\perp,d,(i+1)\, mod N, 0)$, where $s' = s_0 \ldots s_{i-1} d s_{i+1} \ldots s_{N-1}$ and $s'' = s_0 \ldots s_{i-1}\ \perp\ s_{i+1} \ldots s_{N-1}$.
We have $(\mathsf{Fifo}(d), \mathsf{Buff}(s'',\perp,d,(i+1)\, mod N, 0)) \in \mathcal{B}$.

(ii) $\mathsf{Fifo}(\varepsilon) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(\varepsilon)$, where $X \subseteq \mathbb{A}$. By Proposition 3.18.1, $\mathsf{Buff}(s,\perp,\perp,i,0) \xrightarrow{X}_r \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)$, and $(\underline{\mathsf{Fifo}}(\varepsilon), \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)) \in \underline{\mathcal{B}}$.

Consider a generic pair $(\underline{\mathsf{Fifo}}(\varepsilon), \underline{\mathsf{Buff}}(s,\perp,\perp,i,0))$ in $\underline{\mathcal{A}}$ (hence in $\Re$) and the transitions that $\underline{\mathsf{Fifo}}(\varepsilon)$ can perform (see Proposition 3.2). Assume:

(i) $\underline{\mathsf{Fifo}}(\varepsilon) \xrightarrow{in(d)}_r \mathsf{Fifo}(d)$. Again, $s \in \{\perp, \underline{\perp}\}^N$; consider the following transitions $\underline{\mathsf{Buff}}(s,\perp,\perp,i,0) \xrightarrow{in(d)}_r \mathsf{Buff}(\underline{s},d,\perp,i,0) \xrightarrow{\tau}_r \xrightarrow{\tau}_r \mathsf{Buff}(s'',\perp,d,(i+1)\, mod N, 0)$, where $s'' = \underline{s_0} \ldots \underline{s_{i-1}}\ \perp\ \underline{s_{i+1}} \ldots \underline{s_{N-1}} \in \{\perp, \underline{\perp}\}^N$. Then $(\mathsf{Fifo}(d), \mathsf{Buff}(s'',\perp,d,(i+1)\, mod N, 0)) \in \mathcal{B}$.

(ii) $\underline{\mathsf{Fifo}}(\varepsilon) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(\varepsilon)$, where $X \subseteq \mathbb{A} \backslash \{in(0), in(1)\}$. By Proposition 3.18.3 $\underline{\mathsf{Buff}}(s,\perp,\perp,i,0) \xrightarrow{X}_r \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)$ and by hypothesis $(\underline{\mathsf{Fifo}}(\varepsilon), \underline{\mathsf{Buff}}(s,\perp,\perp,i,0)) \in \underline{\mathcal{A}}$.

Consider a generic pair $(\mathsf{Fifo}(at), \mathsf{Buff}(s,\perp,a,i,|t|))$ in $\mathcal{B}$ (hence in $\Re$) and the transitions that $\mathsf{Fifo}(at)$ can perform (see Proposition 3.2). By hypothesis $i \in [0..N]$, $s \in D_\perp^N$, $t \in \mathrm{V}_{N+2}$, $|t| < N$ and $g(s,i,|t|) = t$. Assume:

(i) $\mathsf{Fifo}(at) \xrightarrow{out(a)}_r \mathsf{Fifo}(t)$. Consider the following transition $\mathsf{Buff}(s,\perp,a,i,|t|) \xrightarrow{out(a)}_r \mathsf{Buff}(s,\perp,\perp,i,|t|)$. We distinguish cases $t = \varepsilon$ and $t \neq \varepsilon$:

1. $t = \varepsilon$. By the buff-invariant, $|t| = 0$ implies $s \in \{\perp, \underline{\perp}\}^N$.
   Then $(\mathsf{Fifo}(\varepsilon), \mathsf{Buff}(s,\perp,\perp,i,0)) \in \mathcal{A}$
2. $t \neq \varepsilon$. Then $t = a't'$. Consider the following transition:
   $\mathsf{Buff}(s,\perp,\perp,i,|t|) \xrightarrow{\tau}_r \mathsf{Buff}(s',\perp,a',(i+1)\, mod N, |t|-1)$, where $s_i \in \{a', \underline{a'}\}$ and $s' = s_0 \ldots s_{i-1}\ \perp\ s_{i+1} \ldots s_{N-1}$. By $g(s,i,|t|) = t$, we have $g(s',(i+1)\, mod N, |t|-1) = t'$ and, hence, $(\mathsf{Fifo}(a't'), \mathsf{Buff}(s',\perp,a',(i+1)\, mod N, |t'|)) \in \mathcal{B}$.

(ii) $\mathsf{Fifo}(at) \xrightarrow{in(d)}_r \mathsf{Fifo}(atd)$. We distinguish two cases: $|atd| = N+2$ or $|atd| < N+2$.

1. $|atd| = N+2$. Then consider the following transition:
   $\mathsf{Buff}(s, \bot, a, i, |t|) \xrightarrow{in(d)}_r \mathsf{Buff}(s, d, a, i, |t|)$ and $(\mathsf{Fifo}(atd), \mathsf{Buff}(s, d, a, i, |t|)) \in \mathcal{C}$.
2. $|atd| < N+2$. Hence $|t| < N$. Then consider the following transitions:
   $\mathsf{Buff}(s, \bot, a, i, |t|) \xrightarrow{in(d)}_r \mathsf{Buff}(s, d, a, i, |t|) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, a, i, |t|+1)$,
   where $s' = s_0 \dots s_{j-1} d s_{j+1} \dots s_{N-1}$, for $j = (i + |t|) \, mod \, N$. By $g(s, i, |t|) = t$, we have $g(s', i, |t|+1) = td$, hence $(\mathsf{Fifo}(atd), \mathsf{Buff}(s', \bot, a, i, |t|+1)) \in \mathcal{B}$.

(iii) $\mathsf{Fifo}(at) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(at)$, for every $X \subseteq \mathbb{A}$. Then $\mathsf{Buff}(s, \bot, a, i, |t|) \xrightarrow{X}_r \underline{\mathsf{Buff}}(\underline{s}, \bot, a, i, |t|)$ and $(\underline{\mathsf{Fifo}}(at), \underline{\mathsf{Buff}}(\underline{s}, \bot, a, i, |t|)) \in \underline{\mathcal{B}}$.

Similar reasoning works for pairs $(\underline{\mathsf{Fifo}}(at), \underline{\mathsf{Buff}}(s, \bot, a, i, |t|))$ in $\underline{\mathcal{B}}$ and transitions:

(i) $\underline{\mathsf{Fifo}}(at) \xrightarrow{out(a)}_r \mathsf{Fifo}(t)$. Consider the following transition
$\underline{\mathsf{Buff}}(s, \bot, a, i, |t|) \xrightarrow{out(a)}_r \mathsf{Buff}(\underline{s}, \bot, \bot, i, |t|)$. We distinguish cases $t = \varepsilon$ and $t \neq \varepsilon$:

1. $t = \varepsilon$. By the buff-invariant, $|t| = 0$ implies $\underline{s} = \underline{\bot}^N \in \{\bot, \underline{\bot}\}^N$.
   Then $(\mathsf{Fifo}(\varepsilon), \mathsf{Buff}(\underline{s}, \bot, \bot, i, 0)) \in \mathcal{A}$.
2. $t \neq \varepsilon$. Then $t = a't'$. Consider the following transition:
   $\underline{\mathsf{Buff}}(s, \bot, \bot, i, |t|) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, a', (i+1) \, mod \, N, |t|-1)$, where $s_i \in \{a', \underline{a'}\}$ and $s' = \underline{s_0} \dots \underline{s_{i-1}} \; \bot \; \underline{s_{i+1}} \dots \underline{s_{N-1}}$. By hypothesis $g(s, i, |t|) = t$. Thus $g(s', (i+1) \, mod \, N, |t| - 1) = t'$ and $(\overline{\mathsf{Fifo}(a't')}, \mathsf{Buff}(s', \bot, a', (i+1) \, mod \, N, |t'|)) \in \mathcal{B}$.

(ii) $\underline{\mathsf{Fifo}}(at) \xrightarrow{in(d)}_r \mathsf{Fifo}(atd)$. We distinguish two cases: $|atd| = N+2$ or $|atd| < N+2$.

1. $|atd| = N+2$. Then consider the following transition:
   $\underline{\mathsf{Buff}}(s, \bot, a, i, |t|) \xrightarrow{in(d)}_r \mathsf{Buff}(\underline{s}, d, a, i, |t|)$ and $(\mathsf{Fifo}(atd), \mathsf{Buff}(\underline{s}, d, a, i, |t|)) \in \mathcal{C}$.
2. $|atd| < N+2$. Hence $|t| < N$. Then consider the following transitions:
   $\underline{\mathsf{Buff}}(s, \bot, a, i, |t|) \xrightarrow{in(d)}_r \mathsf{Buff}(\underline{s}, d, a, i, |t|) \xrightarrow{\tau}_r \mathsf{Buff}(s', \bot, a, i, |t|+1)$,
   where $s' = \underline{s_0} \dots \underline{s_{j-1}} d \underline{s_{j+1}} \dots \underline{s_{N-1}}$, for $j = (i+|t|) \, mod \, N$. By hypothesis $g(s, i, |t|) = t$. Thus $g(s', i, |t|+1) = td$ and $(\mathsf{Fifo}(atd), \mathsf{Buff}(s', \bot, a, i, |t|+1)) \in \mathcal{B}$.

(iii) $\underline{\mathsf{Fifo}}(at) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(at)$, for every $X \subseteq \mathbb{A} \backslash \{out(a), in(0), in(1)\}$. Then, by Proposition 3.18.4, $\underline{\mathsf{Buff}}(s, \bot, a, i, |t|) \xrightarrow{X}_r \underline{\mathsf{Buff}}(s, \bot, a, i, |t|)$ and $(\underline{\mathsf{Fifo}}(at), \underline{\mathsf{Buff}}(\underline{s}, \bot, a, i, |t|)) \in \underline{\mathcal{B}}$.

Finally, take a pair $(\mathsf{Fifo}(atd), \mathsf{Buff}(s, d, a, i, N))$ in $\mathcal{C}$. By hypothesis, $i \in [0..N]$, $s \in D_\bot^N$, $t \in \mathrm{V}_{N+2}$, $|t| = N$ and $g(s, i, N) = t$. By Proposition 3.2, $\mathsf{Fifo}(atd)$ can perform the following transitions:

(i) $\mathsf{Fifo}(atd) \xrightarrow{out(a)}_r \mathsf{Fifo}(td)$. Assume $t = a't'$ and consider the following transitions:
$\mathsf{Buff}(s, d, a, i, N) \xrightarrow{out(a)}_r \mathsf{Buff}(s, d, \bot, i, N) \xrightarrow{\tau}_r \mathsf{Buff}(s', d, a', (i+1) \, mod \, N, N-1) \xrightarrow{\tau}_r$
$\mathsf{Buff}(s'', \bot, a', (i+1) \, mod \, N, N)$, where $s_i \in \{a', \underline{a'}\}$, $s' = s_0 \dots s_{i-1} \; \bot \; s_{i+1} \dots s_{N-1}$ and $s'' = s_0 \dots s_{i-1} d s_{i+1} \dots s_{N-1}$; note that $(((i+1) \, mod \, N) + N - 1) \, mod \, N = i$. By definition of g, it is $g(s', (i+1) \, mod \, N, N-1) = t'$ and $g(s'', (i+1) \, mod \, N, N) = t'd$.
Thus, $(\mathsf{Fifo}(a't'd), \mathsf{Buff}(s'', \bot, a', (i+1) \, mod \, N, N)) \in \mathcal{B}$.

(ii) $\mathsf{Fifo}(atd) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(atd)$ for every $X \subseteq \mathbb{A}$. By Proposition 3.18,
$\mathsf{Buff}(s, d, a, i, N) \xrightarrow{X}_r \underline{\mathsf{Buff}}(s, d, a, i, N)$ and $(\underline{\mathsf{Fifo}}(atd), \underline{\mathsf{Buff}}(s, d, a, i, N)) \in \underline{\mathcal{C}}$.

We can prove similar results for pairs $(\underline{\mathsf{Fifo}}(atd), \underline{\mathsf{Buff}}(s,d,a,i,N))$ in $\underline{\mathcal{C}}$. By hypothesis, $i \in [0..N]$, $s \in D_\perp^N$, $t \in \mathrm{V}_{N+2}$, $|t| = N$ and $g(s,i,N) = t$.

By Proposition 3.2, $\underline{\mathsf{Fifo}}(atd)$ can perform the following transitions:

(i) $\underline{\mathsf{Fifo}}(atd) \xrightarrow{out(a)}_r \mathsf{Fifo}(td)$. Assume $t = a't'$ and consider the following transitions:
$\underline{\mathsf{Buff}}(s,d,a,i,N) \xrightarrow{out(a)}_r \mathsf{Buff}(\underline{s},d,\perp,i,N) \xrightarrow{\tau}_r \mathsf{Buff}(s',d,a',(i+1)\,mod\,N, N-1) \xrightarrow{\tau}_r$
$\mathsf{Buff}(s'',\perp,a',(i+1)\,mod\,N,N)$, where $s_i \in \{a', \underline{a'}\}$, $s' = \underline{s_0} \ldots \underline{s_{i-1}} \perp \underline{s_{i+1}} \ldots \underline{s_{N-1}}$ and $s'' = s'_0 \ldots s'_{i-1} d s'_{i+1} \ldots s'_{N-1}$; again, $(((i+1)\,mod\,N) + N - 1)\,mod\,N = i$. By definition of g, it is $g(s',(i+1)\,mod\,N, N-1) = t'$ and $g(s'',(i+1)\,mod\,N,N) = t'$.

Thus $(\mathsf{Fifo}(a't'd), \mathsf{Buff}(s'',\perp,a',(i+1)\,mod\,N,N)) \in \mathcal{B}$.

(ii) $\underline{\mathsf{Fifo}}(atd) \xrightarrow{X}_r \underline{\mathsf{Fifo}}(atd)$ for each $X \subseteq \mathbb{A}\backslash\{out(a)\}$. By Proposition 3.18.7, $\underline{\mathsf{Buff}}(s,d,a,i,N) \xrightarrow{X}_r \underline{\mathsf{Buff}}(s,d,a,i,N)$ and $(\underline{\mathsf{Fifo}}(atd), \underline{\mathsf{Buff}}(s,d,a,i,N)) \in \underline{\mathcal{C}}$.

□

We now give the intuition behind these formal results, which are as one would presumably expect.

- Buff, similarly to Pipe, performs internal activities to manage the store. Hence, it cannot be more efficient than Fifo.
- Although Buff is a distributed implementation (as Pipe is), input and output access the same variables i and m, hence they block each other. Therefore, the effect that prevents Fifo from being more efficient than Pipe does not appear, and Fifo is indeed more efficient than Buff.

4.3 Relating Pipe and Buff

Finally, we relate Pipe and Buff. Again, one would expect that Buff is more efficient, because it takes less time to move an item from input to output, but actually these two buffer implementations are unrelated in general – as the following theorem shows.

Theorem 4.3 1. Pipe $\not\sqsupseteq$ Buff, provided $N > 1$,

2. Pipe $\sqsupseteq$ Buff, provided $N = 1$,

3. Buff $\not\sqsupseteq$ Pipe in any case.

Proof:

1. For $X = \{out(0)\}$, Pipe can perform the refusal trace $v = in(0)\emptyset\emptyset XX$ as follows:
$\mathsf{Pipe}(\perp^{N+2}) \xrightarrow{in(0)}_r \mathsf{Pipe}(\perp^{N+1}\ 0) \xrightarrow{\emptyset}_r \mathsf{Pipe}(\underline{\perp}^{N+1}\ \underline{0}) \xrightarrow{\tau}_r \mathsf{Pipe}(\underline{\perp}^{N}\ 0\ \perp) \xrightarrow{\emptyset}_r$
$\mathsf{Pipe}(\underline{\perp}^{N}\ \underline{0}\ \underline{\perp}) \xrightarrow{\tau}_r \mathsf{Pipe}(\underline{\perp}^{N-1} 0\ \perp\ \underline{\perp}) \xrightarrow{X}_r \mathsf{Pipe}(\underline{\perp}^{N-1}\ \underline{0}\ \underline{\perp}^2) \xrightarrow{\tau}_r \mathsf{Pipe}(\underline{\perp}^{N-2} 0\ \perp\ \underline{\perp}^2) \xrightarrow{X}_r$.
(Note that, in the particular case that $N = 1$, $\mathsf{Pipe}(\perp^3) \xrightarrow{in(0)\emptyset\tau\emptyset\tau}_r \mathsf{Pipe}(0\ \perp\ \underline{\perp}) \xrightarrow{X}_r \mathsf{Pipe}(\underline{0}\ \underline{\perp}^2) \not\xrightarrow{X}_r$; this is the reason why $N > 1$).

Now we prove that Buff cannot perform v; the first step for v would be
$\mathsf{Buff}(\perp^N, \perp, \perp, 0, 0) \xrightarrow{in(0)}_r \mathsf{Buff}(\perp^N, 0, \perp, 0, 0)$. At this point we can either write into Mem or let time pass. Assume first, we perform the write.
$\mathsf{Buff}(\perp^N, 0, \perp, 0, 0) \xrightarrow{\tau}_r \mathsf{Buff}(0\ \perp^{N-1}, \perp, \perp, 0, 1) \xrightarrow{\emptyset}_r \underline{\mathsf{Buff}}(0\ \perp^{N-1}, \perp, \perp, 0, 1) \xrightarrow{\tau}$
$\mathsf{Buff}(\perp\ \underline{\perp}^{N-1}, \perp, 0, 1, 0) \xrightarrow{\emptyset}_r \underline{\mathsf{Buff}}(\perp\ \underline{\perp}^{N-1}, \perp, 0, 1, 0) \not\xrightarrow{X}_r$
If we let time pass instead, then

$\mathsf{Buff}(\bot^N, 0, \bot, 0, 0) \xrightarrow{\emptyset}_r \underline{\mathsf{Buff}}(\bot^N, 0, \bot, 0, 0) \xrightarrow{\tau}_r \mathsf{Buff}(0\underline{\bot}^{N-1}, \bot, \bot, 0, 1) \xrightarrow{\emptyset}_r$
$\underline{\mathsf{Buff}}(0\underline{\bot}^{N-1}, \bot, \bot, 0, 1) \xrightarrow{\tau}_r \mathsf{Buff}(\bot\, \underline{\bot}^{N-1}, \bot, 0, 1, 0) \xrightarrow{X}_r$
$\underline{\mathsf{Buff}}(\bot\, \underline{\bot}^{N-1}, \bot, 0, 1, 0) \not\xrightarrow{X}_r$. In both cases, Buff cannot perform v, hence $v \notin \mathsf{RT}(\mathsf{Buff})$ and we are done.

2. We show that $\mathsf{RT}(\mathsf{Pipe}(\bot^3)) \subseteq \mathsf{RT}(\mathsf{Buff}(\bot, \bot, \bot, 0, 0))$ by giving a simulation relation containing $(\mathsf{Pipe}(\bot^3)), \mathsf{Buff}(\bot, \bot, \bot, 0, 0))$.

   Consider the set of pairs
   $(\mathsf{Pipe}(s_0 s_1 s_2), \mathsf{Buff}(s_1, x_2, x_0, 0, m))$ and
   $(\mathsf{Pipe}(\underline{x_0 x_1 x_2}), \underline{\mathsf{Buff}}(s_1, x_2, x_0, 0, m))$, where $s_i \in D_\bot$ and $x_i \in \mathrm{V} \cup \{\bot\}$ are such that $s_i \in \{x_i, \underline{x_i}\}$ and either $s_1 \in \{\bot, \underline{\bot}\}$ and $m = 0$ or $s_1 \notin \{\bot, \underline{\bot}\}$ and $m = 1$. This is a simulation relation containing $(\mathsf{Pipe}(\bot^3)), \mathsf{Buff}(\bot, \bot, \bot, 0, 0))$.

3. If we had $\mathsf{Buff}(\bot^N, \bot, \bot, i, 0) \sqsupseteq \mathsf{Pipe}(\bot^{N+2})$ then, by Theorem 4.2.2 and transitivity of $\sqsupseteq$, we would have $\mathsf{Fifo} \sqsupseteq \mathsf{Pipe}$. This would contradict Theorem 4.1.2.

□

We now give the intuition behind these formal results.

- Buff, similarly to Pipe, performs internal activities to manage the store. The number of such internal actions for one item is 2 in Buff, namely one for putting the item into the array and one for taking it out. For Pipe, this number is larger in case $N > 1$, explaining why Pipe cannot be more efficient than Buff.
- For $N = 1$, the array of Buff degenerates to a cell for one item; hence, Pipe and Buff have a very similar architecture. But in Pipe, input and output are independent, while in Buff they block each other, as explained above. Thus, Pipe is faster than Buff in this case.
- This independence of input and output in Pipe always prevents Buff from being more efficient.

References

[1] S. Arun-Kumar and M. Hennessy. An efficiency preorder for processes. *Acta Informatica* 29:737–760, 1992.

[2] E. Bihler and W. Vogler. Efficiency of Token-Passing MUTEX-Solutions – Some Experiments. In Applications and Theory of Petri Nets 1998, LNCS 1420, pp. 185–204, 1998.

[3] F. Corradini, W. Vogler and L. Jenner. Comparing the Worst-Case Efficiency of Asynchronous Systems with PAFAS. Internal Report n.31-2000, University of L'Aquila, 2000. Available from: http://w3.dm.univaq.it/∼flavio.

[4] R. De Nicola and M.C.B. Hennessy. Testing equivalence for processes. *Theoret. Comput. Sci.*, 34:83–133, 1984.

[5] L. Jenner and W. Vogler. Fast asynchronous systems in dense time. In Proc. of ICALP'96, LNCS 1099, pp. 75–86, 1996. To appear in Theoret. Comput. Sci.

[6] L. Jenner and W. Vogler. Comparing the Efficiency of Asynchronous Systems. In Proc. of AMAST Workshop on Real-Time and Probabilistic Systems, LNCS 1601, pp. 172–191, 1999.

[7] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[8] W. Vogler. Faster asynchronous systems. In Proc. of CONCUR'95, LNCS 962, pp. 299-312, 1995.