Read Operators and their Expressiveness in Process Algebras[*]

Flavio Corradini   Maria Rita Di Berardini
School of Science and Technology, Computer Science Division,
University of Camerino
flavio.corradini@unicam.it, mariarita.diberardini@unicam.it

Walter Vogler
Institut für Informatik,
Universität Augsburg, Germany
vogler@informatik.uni-augsburg.de

We study two different ways to enhance PAFAS, a process algebra for modelling asynchronous timed concurrent systems, with non-blocking reading actions. We first add reading in the form of a read-action prefix operator. This operator is very flexible, but its somewhat complex semantics requires two types of transition relations. We also present a read-set prefix operator with a simpler semantics, but with syntactic restrictions. We discuss the expressiveness of read prefixes; in particular, we compare them to read-arcs in Petri nets and justify the simple semantics of the second variant by showing that its processes can be translated into processes of the first with timed-bisimilar behaviour. It is still an open problem whether the first algebra is more expressive than the second; we give a number of laws that are interesting in their own right, and can help to find a backward translation.

1 Introduction

Non-blocking reading is an important feature e.g. for proving the liveness of MUTEX solutions under the progress assumption (aka weak fairness). We study the first process algebra with non-blocking read actions, where 'read' refers to accessing a variable, e.g. modelled as a separate process *Var*. Observe that read is an activity of *Var*, and in a setting with explicit modelling of data, it would rather be an output than an input action of *Var*.

Non-blocking reading is known from Petri nets, where it has been added in the form of read arcs; these allow multiple concurrent reading of the same resource, a quite frequent situation in many distributed systems. Read arcs represent *positive context conditions*, i.e. elements which are needed for an event to occur, but are not affected by it. As argued in [17], the importance of such elements is twofold. Firstly, they allow a faithful representation of systems where the notion of "reading without consuming" is commonly used, like database systems [20] or any computation framework based on shared memory. Secondly, they allow to specify directly and naturally a level of concurrency greater than in classical nets: two transitions reading the same place may also occur simultaneously; in classical nets, the transitions would be connected to the place by loops (namely, i.e. reading is modelled through a "rewrite" operation) which does not allow the simultaneous execution of two tasks that read the same resource. Read arcs have been used to model a variety of applications such as transaction serialisability in databases [20], concurrent constraint programming [18], asynchronous systems [22], and cryptographic protocols [14]. Reading is also related to the notion of *persistence* e.g. in several calculi for describing and analysing security protocols; in particular, persistent messages (that can be read but not consumed) are used to model that every message can be remembered by the spy (see [4] and the references therein).

[*]This work was supported by the PRIN Project 'Paco: Performability-Aware Computing: Logics, Models, and Languages

Semantics and expressivity of read arcs have been studied e.g. in the following: [5] discusses a step semantics; [2] shows that timed Petri nets with read arcs unify timed Petri nets and timed automata. Finally, [22] shows that read arcs add relevant expressivity; the MUTEX problem can be solved with nets having read arcs but not with ordinary nets having no read arcs.

In this paper, we present two different ways to enhance PAFAS [11], a process algebra for modelling asynchronous timed concurrent systems, with non-blocking reading actions. PAFAS was introduced for evaluating the worst-case efficiency of asynchronous systems. It was also used in [7, 8] for studying (weak) fairness of actions and components in system computations, similarly to results of [22] for a Petri net setting. This fairness requires that an action has to be performed (a component has to perform an action, resp.), whenever it is enabled continuously in a run. Fairness can be defined in an intuitive but complicated way in the spirit of [13, 12], and we proved that each everlasting (or non-Zeno maximal) timed run is fair and vice versa [7]. We used these characterisations in [8] to prove that Dekker's MUTEX algorithm satisfies the respective liveness property under the assumption of *fairness of components*, while this fails under *fairness of actions*. To improve this, one needs suitable assumptions about the hardware, cf. [19], namely that reading a value from a storage cell is non-blocking; to model this we introduce specific reading prefixes for PAFAS.

We first add reading in the form of a read-action prefix $\alpha \triangleright Q$ (the new process language is called PAFAS$_r$), which behaves as Q but, like a variable or a more complex data structure, can also be read with the action α. Since being read should not change the state, α can be repeated until the execution of some ordinary action of Q. Thus, e.g. $a \triangleright b.\mathsf{nil}$ can perform any number of a's until it terminates via an ordinary b. The operational semantics for $\alpha \triangleright Q$ needs two types of transition relations to properly deal e.g. with sequences of read actions.

Under some syntactic restrictions, the semantics can be simplified. To be still able to express sequences of read actions directly, we introduced a read-set operator $\{a_1, \cdots, a_n\} \triangleright Q$ in the language PAFAS$_s$. In [9], we already used PAFAS$_s$ to show the correctness of Dekker's algorithm: regarding some actions as reading, this algorithm satisfies MUTEX liveness already under the assumption of *fairness of actions*. It had long been an open problem how to achieve such a result in a process algebra [23]. The simpler semantics of PAFAS$_s$ is helpful for building tools. Indeed, we have already proved some MUTEX algorithms correct or incorrect with the aid of the automated verification tool `FASE` [3]. We plan to continue this work by also considering the efficiency of MUTEX algorithms and other systems.

In this paper, we study PAFAS$_r$ and PAFAS$_s$ further with special attention to expressiveness. The first issue is that PAFAS$_r$ models non-blocking reading in an intuitive way, while the necessary restrictions in case of PAFAS$_s$ are not so obvious. In fact, the investigations for this paper have disclosed that the restrictions in [9] still allowed processes with a contra-intuitive semantics. To rectify this subtle mistake, we give an improved definition of *proper* PAFAS$_s$ processes[1], and we show how to translate each proper process Q into a PAFAS$_r$ process whose timed behaviour is bisimilar and even isomorphic to that of Q. This shows at the same time that a proper process really has an intuitive behaviour and that PAFAS$_r$ is at least as expressive as the proper fragment of PAFAS$_s$.

In this paper, we additionally show that safe Petri nets with read-arcs as in [22] can be modelled with proper PAFAS$_s$ processes. It is still an open problem whether PAFAS$_r$ is more expressive than PAFAS$_s$; we present a number of laws that are interesting in their own right and give a backward translation for a fragment of PAFAS$_r$. Constructing a general backward translation seems to be related to finding an expansion law for PAFAS$_r$ processes, a law that is not even known for standard PAFAS processes.

We have also extended the correspondence between fair and everlasting runs; thus, also in PAFAS$_r$

[1]Luckily, the model of Dekker's algorithm in [9] is also proper as defined here.

and in PAFAS$_s$, we capture fairness with timed behaviour. To demonstrate the extended expressiveness of reading with a concrete example, we prove that no finite-state process in standard PAFAS has the same fair language as $a \triangleright b.\mathsf{nil}$ (Theorem 2.5).

The rest of the paper is organised as follows. Sections 2 and 3 introduce PAFAS$_r$ and PAFAS$_s$ with their respective timed operational semantics and prove result regarding $a \triangleright b$. Section 4 provides a mapping from PAFAS$_s$ to PAFAS$_r$ and presents the result for Petri nets. The backward translation is discussed in Section 5. Finally, Section 6 presents some concluding remarks. Some proofs can be found in the appendices.

2 A process algebra for describing read behaviours

In this section, we introduce PAFAS$_r$ and give a first expressiveness result. PAFAS is a CCS-like process description language [16] (with a *TCSP*-like parallel composition [1]), where actions are atomic and instantaneous but have associated an upper time bound (either 0 or 1, for simplicity) interpreted as a maximal time delay for their execution. As explained in [11], these upper time bounds can be used for evaluating the performance of asynchronous systems, but do not influence *functionality* (which actions are performed); so compared to CCS, also PAFAS treats the full functionality of asynchronous systems. W.r.t. the original language, here we introduce the new *read prefix* $\triangleright$ to represent non-blocking behaviour of processes. Intuitively, the term $\alpha \triangleright P$ models a process like a variable or a more complex data structure that behaves as P but can additionally be read with α: since being read does not change the state, α can be performed repeatedly until the execution of some ordinary action of P, and it does not block a synchronisation partner (a reading process) as described below.

We use the following notation. $\mathbb{A}$ is an infinite set of *visible actions*. An additional action τ is used to represent internal activity, which 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 $\mathbb{A}_\tau$ by $\alpha, \beta, \ldots$. Actions in $\mathbb{A}_\tau$ can let time 1 pass before their execution, i.e. 1 is their maximal delay. After that time, they become *urgent* actions written $\underline{a}$ or $\underline{\tau}$; these cannot be delayed. 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 $\underline{\alpha}, \underline{\beta}, \ldots$. Elements of $\mathbb{A}_\tau \cup \underline{\mathbb{A}}_\tau$ are ranged over by μ. $\mathscr{X}$ (ranged over by $x, y, z, \ldots$) is the set of process variables, used for recursive definitions. $\Phi : \mathbb{A}_\tau \to \mathbb{A}_\tau$ is a *general relabelling function* if the set $\{\alpha \in \mathbb{A}_\tau \mid \emptyset \neq \Phi^{-1}(\alpha) \neq \{\alpha\}\}$ is finite and $\Phi(\tau) = \tau$. Such a function can also be used to define *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 $\alpha \notin A$.

We assume that time elapses in a discrete way[2]. Thus, an action prefixed process $a.P$ can either do action a and become process P (as usual in CCS) or can let one time step pass and become $\underline{a}.P$; $\underline{a}$ is called *urgent* a, and $\underline{a}.P$ cannot delay a, but *as a stand-alone process* can only do a to become P. In the following, initial processes are just processes of a standard process algebra extended with $\triangleright$. General processes include all processes reachable from the initial ones according to the operational semantics to be defined below.

The sets $\tilde{\mathbb{P}}_1$ of *initial (timed) process terms P* and $\tilde{\mathbb{P}}$ of (general) *(timed) process terms Q* is generated by the following grammar:

[2] PAFAS is not time domain dependent, meaning that the choice of discrete or continuous time makes no difference for the testing-based semantics of asynchronous systems, see [11] for more details.

$$\begin{array}{rcl} P & ::= & \mathsf{nil} \mid x \mid \alpha.P \mid \alpha \triangleright P \mid P+P \mid P \,\|_A\, P \mid P[\Phi] \mid \mathsf{rec}\, x.P \\ Q & ::= & P \mid \underline{\alpha}.P \mid \mu \triangleright Q \mid Q+Q \mid Q \,\|_A\, Q \mid Q[\Phi] \mid \mathsf{rec}\, x.Q \end{array}$$

where nil is a constant, $x \in \mathscr{X}$, $\alpha \in \mathbb{A}_\tau$, $\mu \in \mathbb{A}_\tau \cup \underline{\mathbb{A}}_\tau$, Φ is a general relabelling function and $A \subseteq \mathbb{A}$ possibly infinite. We say that a variable $x \in \mathscr{X}$ is *guarded* in Q if it only appears in the scope of some $\mu \in \mathbb{A}_\tau \cup \underline{\mathbb{A}}_\tau$. We assume that recursion is *guarded*, i.e. for $\mathsf{rec}\, x.Q$ variable x is guarded in Q. A process term is *closed* if every variable x is bound by the corresponding $\mathsf{rec}\, x$-operator; the set of closed timed process terms in $\tilde{\mathbb{P}}$ and $\tilde{\mathbb{P}}_1$, simply called *processes* and *initial processes* resp., is denoted by $\mathbb{P}$ and $\mathbb{P}_1$ resp.

We briefly describe the operators. The nil-process cannot perform any action, but may let time pass without limit. A trailing nil will often be omitted, so e.g. $a.b+c$ abbreviates $a.b.\mathsf{nil}+c.\mathsf{nil}$. $\mu.Q$ is (action-)prefixing known from CCS. Read-prefixed terms $\alpha \triangleright Q$ and $\underline{\alpha} \triangleright Q$ behave like Q except for the (lazy and urgent, resp.) non-blocking action α. In both cases α is always enabled until component Q evolves via some ordinary action; moreover, $\underline{\alpha}$ stays urgent even if it is performed. $Q_1 + Q_2$ models the choice between processes Q_1 and Q_2. $Q_1 \,\|_A\, Q_2$ is the parallel composition of two processes Q_1 and Q_2 that run in parallel and have to synchronise on all actions from A; this synchronisation discipline is inspired from *TCSP*. $Q[\Phi]$ behaves as Q but with the actions changed according to Φ. $\mathsf{rec}\, x.Q$ models a recursive definition. We often use equations to define recursive processes, e.g. $P \Leftarrow a.P+b$; in contrast, $\equiv$ stands for syntactically equal. Below we use the (syntactic) sort of a process that contains all visible actions the process can ever perform.

Definition 2.1 (*sort*) For a general relabelling function Φ let $ib(\Phi) = \{a \in \mathbb{A} \mid \emptyset \neq \Phi^{-1}(a) \neq \{a\}\}$ (the image base of Φ); by definition of a general relabelling function, $ib(\Phi)$ is finite. The *sort* of $Q \in \tilde{\mathbb{P}}$ is the set $\mathscr{L}(Q) = \{a \in \mathbb{A} \mid a \text{ occurs in } Q\} \cup \bigcup_{\Phi \text{ occurs in } Q} ib(\Phi)$.

The transitional semantics describing the functional behaviour of PAFAS$_r$ terms indicates which actions they can perform. We need two different transition relations $\overset{\alpha}{\mapsto}$ and $\overset{\alpha}{\rightsquigarrow}$ to describe, resp., the ordinary and the reading behaviour of PAFAS$_r$ processes. The functional behaviour is the union of these two kinds of behaviour.

Definition 2.2 (*functional operational semantics*) Let $Q \in \tilde{\mathbb{P}}$ and $\alpha \in \mathbb{A}_\tau$. We say that $Q \xrightarrow{\alpha} Q'$ if $Q \overset{\alpha}{\mapsto} Q'$ or $Q \overset{\alpha}{\rightsquigarrow} Q'$, where the SOS-rules defining the transition relations $\overset{\alpha}{\mapsto} \subseteq (\tilde{\mathbb{P}} \times \tilde{\mathbb{P}})$ (the *ordinary action transitions*) and $\overset{\alpha}{\rightsquigarrow} \subseteq (\tilde{\mathbb{P}} \times \tilde{\mathbb{P}})$ (the *read action transitions*) for $\alpha \in \mathbb{A}_\tau$, are given in Tables 1 and 2, resp.[3]. As usual, we write $Q \xrightarrow{\alpha} Q'$ if $(Q, Q') \in \xrightarrow{\alpha}$ and $Q \xrightarrow{\alpha}$ if $Q \xrightarrow{\alpha} Q'$ for some $Q' \in \tilde{\mathbb{P}}$; and analogously for other types of transition relations.

Rule PREF$_o$ in Table 1 describes the behaviour of an action-prefixed process as usual in CCS. Note that timing can be disregarded: when an action is performed, one cannot see whether it was urgent or not, and thus $\underline{\alpha}.P \overset{\alpha}{\mapsto} P$; furthermore, $\alpha.P$ has to act *within* time 1, i.e. it can also act immediately, giving $\alpha.P \overset{\alpha}{\mapsto} P$. Rule READ$_o$ says that $\mu \triangleright Q$ performs the same ordinary actions as Q removing the read-prefix at the same time. Note that in rule PAR$_{o_2}$, an ordinary action transition can synchronise with both an ordinary and a *read action* transition. The other rules are as expected. Symmetric rules have been omitted.

[3]We do here without functions clean and unmark, used e.g. in [7] to get a closer relationship between states of untimed fair runs and timed non-Zeno runs. They do not change the behaviour (up to an injective bisimulation) and would complicate the setting.

$$\textsc{Pref}_o\ \frac{\mu \in \{\alpha, \underline{\alpha}\}}{\mu.P \overset{\alpha}{\mapsto} P} \qquad \textsc{Read}_o\ \frac{Q \overset{\alpha}{\mapsto} Q'}{\mu \triangleright Q \overset{\alpha}{\mapsto} Q'} \qquad \textsc{Sum}_o\ \frac{Q_1 \overset{\alpha}{\mapsto} Q'}{Q_1 + Q_2 \overset{\alpha}{\mapsto} Q'}$$

$$\textsc{Par}_{o1}\frac{\alpha \notin A,\ Q_1 \overset{\alpha}{\mapsto} Q_1'}{Q_1\|_A Q_2 \overset{\alpha}{\mapsto} Q_1'\|_A Q_2} \qquad \textsc{Par}_{o2}\frac{\alpha \in A,\ Q_1 \overset{\alpha}{\mapsto} Q_1',\ Q_2 \xrightarrow{\alpha} Q_2'}{Q_1\|_A Q_2 \overset{\alpha}{\mapsto} Q_1'\|_A Q_2'}$$

$$\textsc{Rel}_o\frac{Q \overset{\alpha}{\mapsto} Q'}{Q[\Phi] \overset{\Phi(\alpha)}{\mapsto} Q'[\Phi]} \qquad \textsc{Rec}_o\frac{Q\{\mathsf{rec}\,x.Q/x\} \overset{\alpha}{\mapsto} Q'}{\mathsf{rec}\,x.Q \overset{\alpha}{\mapsto} Q'}$$

Table 1: Ordinary behaviour of PAFAS$_r$ processes

$$\textsc{Read}_{r1}\ \frac{\mu \in \{\alpha, \underline{\alpha}\}}{\mu \triangleright Q \overset{\alpha}{\rightsquigarrow} \mu \triangleright Q} \qquad \textsc{Read}_{r2}\ \frac{Q \overset{\alpha}{\rightsquigarrow} Q'}{\mu \triangleright Q \overset{\alpha}{\rightsquigarrow} \mu \triangleright Q'} \qquad \textsc{Sum}_r\ \frac{Q_1 \overset{\alpha}{\rightsquigarrow} Q_1'}{Q_1 + Q_2 \overset{\alpha}{\rightsquigarrow} Q_1' + Q_2}$$

$$\textsc{Par}_{r1}\frac{\alpha \notin A,\ Q_1 \overset{\alpha}{\rightsquigarrow} Q_1'}{Q_1\|_A Q_2 \overset{\alpha}{\rightsquigarrow} Q_1'\|_A Q_2} \qquad \textsc{Par}_{r2}\frac{\alpha \in A,\ Q_1 \overset{\alpha}{\rightsquigarrow} Q_1',\ Q_2 \overset{\alpha}{\rightsquigarrow} Q_2'}{Q_1\|_A Q_2 \overset{\alpha}{\rightsquigarrow} Q_1'\|_A Q_2'}$$

$$\textsc{Rel}_r\frac{Q \overset{\alpha}{\rightsquigarrow} Q'}{Q[\Phi] \overset{\Phi(\alpha)}{\rightsquigarrow} Q'[\Phi]} \qquad \textsc{Rec}_r\frac{Q\{\mathsf{rec}\,x.Q/x\} \overset{\alpha}{\rightsquigarrow} Q'}{\mathsf{rec}\,x.Q \overset{\alpha}{\rightsquigarrow} Q'}$$

Table 2: Reading Behaviour of PAFAS$_r$ processes

Most of the rules in Table 2 say that the execution of reading actions does not change the state of a term Q. Rule $\textsc{Read}_{r2}$ is crucial to manage arbitrarily nested reading actions; contrast it with $\textsc{Read}_o$. Due to technical reasons, rule $\textsc{Rec}_r$ allows unfolding of recursive terms; thus e.g. $\mathsf{rec}\,x.\,a \triangleright b.x \overset{a}{\rightsquigarrow} a \triangleright b.(\mathsf{rec}\,x.\,a \triangleright b.x)$. Notice that this leads to a timed bisimilar process, cf. Section 4.

To give SOS-rules for the time steps of process terms, we consider (partial) *time-steps* like $Q \xrightarrow{X}_r Q'$ where the set $X \subseteq \mathbb{A}$ (called a *refusal set*) consists of non-urgent actions. Hence Q is justified in delaying, i.e. refusing them; Q can take part in a real time step only if it has to synchronise on its urgent actions, and these are delayed by the environment. If $X = \mathbb{A}$ then Q is fully justified in performing this full unit-time step; i.e., Q can perform it independently of the environment. If $Q \xrightarrow{\mathbb{A}}_r Q'$, we write $Q \xrightarrow{1} Q'$; we say that Q performs a *1-step*.

Definition 2.3 (*refusal transitional semantics*) The inference rules in Table 3 define $\xrightarrow{X}_r \subseteq \tilde{\mathbb{P}} \times \tilde{\mathbb{P}}$ where $X \subseteq \mathbb{A}$. A refusal trace of a term $Q \in \tilde{\mathbb{P}}$ records from a *run* of Q which visible actions are performed ($Q \xrightarrow{a} Q'$, $a \in \mathbb{A}$) and which actions Q refuses to perform when time elapses ($Q \xrightarrow{X}_r Q'$, $X \subseteq \mathbb{A}$); i.e. a refusal trace of Q is the sequence of actions from $\mathbb{A}$ and refusal sets $\subseteq \mathbb{A}$ occurring in a finite transition sequence from Q (abstracting from τ-transitions).

Rule $\textsc{Pref}_{t_1}$ says that $\alpha.P$ can let time pass and refuse to perform any action while rule $\textsc{Pref}_{t_2}$ says that $\underline{\alpha}.P$ can let time pass in an appropriate context, but cannot refuse the action α. Process $\underline{\tau}.P$ cannot let time pass at all since, in any context, $\underline{\tau}.P$ has to perform τ before time can pass further. Rule $\textsc{Par}_t$ defines which actions a parallel composition can refuse during a time-step. $Q_1\|_A Q_2$ can refuse the action α if either $\alpha \notin A$ and α can be refused by both Q_1 and Q_2 or $\alpha \in A$ and at least one of Q_1 and Q_2 can

$$\textsc{Nil}_t\ \frac{}{\mathsf{nil} \xrightarrow{X}_r \mathsf{nil}} \qquad \textsc{Pref}_{t1}\ \frac{}{\alpha.P \xrightarrow{X}_r \underline{\alpha}.P} \qquad \textsc{Pref}_{t2}\ \frac{\alpha \notin X \cup \{\tau\}}{\underline{\alpha}.P \xrightarrow{X}_r \underline{\alpha}.P}$$

$$\textsc{Read}_{t1}\ \frac{Q \xrightarrow{X}_r Q'}{\alpha \triangleright Q \xrightarrow{X}_r \underline{\alpha} \triangleright Q'} \qquad \textsc{Read}_{t2}\ \frac{Q \xrightarrow{X}_r Q',\ \alpha \notin X \cup \{\tau\}}{\underline{\alpha} \triangleright Q \xrightarrow{X}_r \underline{\alpha} \triangleright Q'}$$

$$\textsc{Sum}_t\ \frac{Q_i \xrightarrow{X}_r Q_i' \text{ for } i = 1,2}{Q_1 + Q_2 \xrightarrow{X}_r Q_1' + Q_2'} \qquad \textsc{Rel}_t\ \frac{Q \xrightarrow{\Phi^{-1}(X \cup \{\tau\}) \setminus \{\tau\}}_r Q'}{Q[\Phi] \xrightarrow{X}_r Q'[\Phi]}$$

$$\textsc{Rec}_t\ \frac{Q\{\mathsf{rec}\,x.Q/x\} \xrightarrow{X}_r Q'}{\mathsf{rec}\,x.Q \xrightarrow{X}_r Q'} \qquad \textsc{Par}_t\ \frac{Q_i \xrightarrow{X_i}_r Q_i' \text{ for } i = 1,2, X \subseteq (A \cap (X_1 \cup X_2)) \cup ((X_1 \cap X_2) \setminus A)}{Q_1 \|_A Q_2 \xrightarrow{X}_r Q_1' \|_A Q_2'}$$

Table 3: Refusal transitional semantics of PAFAS$_r$ processes

delay it, forcing the other Q_i to wait. Thus, an action is urgent (cannot be further delayed) only when all synchronising 'local' actions are urgent. The other rules are as expected.

Example 2.4 As an example for the definitions given so far, consider an *array* with two Boolean values t and f and define its behaviour as $B_{tf} \equiv P_t \|_A Q_f$ where $P_t \Leftarrow r_{tt} \triangleright (r_t^1 \triangleright w_f^1.P_f) + r_{tf} \triangleright (r_t^1 \triangleright w_f^1.P_f)$, $Q_f \Leftarrow r_{tf} \triangleright (r_f^2 \triangleright w_t^2.Q_t) + r_{ff} \triangleright (r_f^2 \triangleright w_t^2.Q_t)$ and $A = \{r_{ij} \,|\, i,j \in \{t,f\}\}$. Actions r_{ij}, where $i,j \in \{t,f\}$, allow reading both entries at the same time, while r_j^k and w_j^k represent, resp., the reading and the writing of the value $j \in \{t,f\}$ for the entry $k \in \{1,2\}$. By rules $\textsc{Read}_{r1}$ and $\textsc{Read}_{r2}$, $B_{tf} \overset{r_{tf}}{\rightsquigarrow} B_{tf}$ and $B_{tf} \overset{r_t^1}{\rightsquigarrow} B_{tf}$ describing non-blocking reading. P_t offers a choice between r_{tf} and r_{tt}, where synchronisation disallows the latter. Performing w_f^1 after a 1-step does not change the second component, so r_f^2 is still urgent; this shows that w_f^1 does not block r_f^2. With just one type of action transition, P_t would lose the prefix $r_{tf}\triangleright$ when performing r_t^1. Only the execution of an ordinary action can change the state of the array, e.g. $B_{tf} \overset{w_f^1}{\mapsto} B_{ff} \equiv P_f \|_A Q_f$ by Rule $\textsc{Read}_o$.

In [11], it is shown that inclusion of refusal traces characterises an efficiency preorder which is intuitively justified by a testing scenario. In this sense, e.g. $P \equiv a \triangleright b$ is faster than the functionally equivalent $Q \equiv \mathsf{rec}\,x.\,(a.x + b)$, since only the latter has the refusal traces $1a(1a)^*$: after $1a$, Q returns to itself, since recursion unfolding creates fresh a and b; intuitively, b is disabled during the occurrence of a, so a and also b can be delayed again. In contrast, after a time step and any number of as, P turns into $\underline{a} \triangleright \underline{b}$ and no further 1-step is possible. Since read actions do not block or delay other activities, they make processes faster and, hence, have an impact on timed behaviour of systems. If a models the reading of a value stored by P or Q and two parallel processes want to read it, this should take at most time 1 in a setting with non-blocking reads. And indeed, whereas $Q \|_{\{a\}} (a \|_\emptyset a)$ has the refusal trace $1a1a$, this behaviour is not possible for $P \|_{\{a\}} (a \|_\emptyset a)$. Thus, P offers a *faster* service.

Another application of refusal traces is the modelling of *weak fairness of actions*. Weak fairness requires that an action must be performed whenever continuously enabled in a run. Thus, a run from P with infinitely many a's is not fair; the read action does not block b or change the state, so the same b is always enabled but never performed. In contrast, if Q performs a, a fresh b is created; in conformance to [12], a run with infinitely many a's is fair. In [10], generalising [7], fair traces for PAFAS$_r$ (and PAFAS$_s$) are first defined in an intuitive, but very complex fashion in the spirit of [12] and then characterised: they are the sequences of visible actions occurring in transition sequences with infinitely many 1-steps[4]. Due to lack of space, we cannot properly formulate this as a theorem, but take it as a (time-based) **definition** of *fair traces* instead; $\mathsf{FairL}(R)$ is the set of fair traces of R. With this, infinitely many a's are a fair trace of Q since it can repeat $1a$ indefinitely, but the fair traces of finite-state P are those that end with b. This shows an added expressivity of read prefixes:

Theorem 2.5 *If $R \in \tilde{\mathbb{P}}$ is a finite-state process without read-prefixes and with sort $\mathscr{L}(R) = \{a,b\}$, then* $\mathsf{FairL}(R) \neq \{a^i b \,|\, i \geq 0\} = \mathsf{FairL}(a \triangleright b)$.

We can view fairness as imposing a kind of priority for b in P since, in contrast to a, it must be executed in a fair trace. This is of course very different from the usual treatment of priorities [6], since a can be prefered to b for a number of times. The following example shows that read actions can model more than two levels of priority.

Example 2.6 In $P \equiv a \triangleright ((\mathsf{rec}\, x.b.x) \parallel_{\{b\}} b \triangleright c)$, there are three levels of priority: in a fair trace we can perform arbitrarily many a's while both b and c remain enabled and have priority – so far, we can have at most one 1-step. If b occurs, the action a disappears but we can perform arbitrarily many b's while c remains enabled and has priority – with, still, at most one 1-step. Formally, with a 1-step P evolves into $\underline{P} \equiv \underline{a} \triangleright (\underline{b}.(\mathsf{rec}\, x.b.x) \parallel_{\{b\}} \underline{b} \triangleright \underline{c})$. $\underline{P}$ can perform an a to itself, a c (and become $\underline{b}.(\mathsf{rec}\, x.b.x) \parallel_{\{b\}} \mathsf{nil}$), or repeated b's to $((\mathsf{rec}\, x.b.x) \parallel_{\{b\}} \underline{b} \triangleright \underline{c}$; no further 1-steps are possible due to the urgent c; so in a fair trace, finally c is performed to $((\mathsf{rec}\, x.b.x) \parallel_{\{b\}} \mathsf{nil})$ – where infinitely many 1-steps are possible.

3 A read operator with a simpler semantics

The special reading transitions of PAFAS$_r$ are needed to properly derive e.g. $P \equiv a \triangleright b \triangleright Q \xrightarrow{b} a \triangleright b \triangleright Q$. To get a simpler semantics, the idea is to collect all enabled reading actions of a ‘sequential component’ in a set and write e.g. P as $\{a,b\} \triangleright c$. Thus, we define a new kind of read operator $\{\mu_1, \ldots, \mu_n\} \triangleright Q$ with a slightly different syntax. In this way we try to avoid terms with nested reading actions and, as a consequence, we can describe the behaviour of the new PAFAS$_s$ processes by means of a simpler timed operational semantics with just one type of action transitions. A price to pay is that not all PAFAS$_s$ processes have a reasonable semantics; but the subset with a reasonable semantics is practically expressive enough (e.g. for expressing MUTEX solutions adequately) due to the *set* of reading actions, cf. [9].

The sets $\tilde{\mathbb{S}}_1$ of *initial (timed) process terms P* and $\tilde{\mathbb{S}}$ of (general) *(timed) process terms Q* is generated by the following grammar:

$$\begin{array}{rcl} P & ::= & \mathsf{nil} \,\big|\, x \,\big|\, \alpha.P \,\big|\, \{\alpha_1, \ldots, \alpha_n\} \triangleright P \,\big|\, P + P \,\big|\, P \parallel_A P \,\big|\, P[\Phi] \,\big|\, \mathsf{rec}\, x.P \\ Q & ::= & P \,\big|\, \underline{\alpha}.P \,\big|\, \{\mu_1, \ldots, \mu_n\} \triangleright Q \,\big|\, Q + Q \,\big|\, Q \parallel_A Q \,\big|\, Q[\Phi] \,\big|\, \mathsf{rec}\, x.Q \end{array}$$

where nil is a constant, $x \in \mathscr{X}$, $\alpha \in \mathbb{A}_\tau$, $\{\alpha_1, \ldots, \alpha_n\} \subseteq \mathbb{A}_\tau$ finite, $\{\mu_1, \ldots, \mu_n\}$ is a finite subset of $\mathbb{A}_\tau \cup \underline{\mathbb{A}}_\tau$ that cannot contain two copies (one lazy and the other one urgent) of the same action α, i.e. $\big|\{\alpha, \underline{\alpha}\} \cap \{\mu_1, \ldots, \mu_n\}\big| \leq 1$ for any $\alpha \in \mathbb{A}_\tau$. Again, Φ is a general relabelling function and $A \subseteq \mathbb{A}$ possibly infinite. Also in this section, recursion is guarded. The sets of closed timed process terms in $\tilde{\mathbb{S}}$ and $\tilde{\mathbb{S}}_1$, simply called *processes* and *initial processes* resp., are $\mathbb{S}$ and $\mathbb{S}_1$ resp.

[4]Observe that [9] just contains the application presented in [10]; PAFAS$_r$ is not treated there at all.

Definition 3.1 (*functional operational semantics*) The SOS-rules defining the transition relations $\xrightarrow{\alpha} \subseteq (\tilde{\mathbb{S}} \times \tilde{\mathbb{S}})$ (the *action transitions*) are those in Table 1[5] where we replace the rule $\textsc{Read}_o$ with:

$$\textsc{Read}_{s1}\ \frac{\mu_i \in \{\alpha, \underline{\alpha}\}}{\{\mu_1,\dots,\mu_n\} \triangleright Q \xrightarrow{\alpha} \{\mu_1,\dots,\mu_n\} \triangleright Q} \qquad \textsc{Read}_{s2}\ \frac{Q \xrightarrow{\alpha} Q'}{\{\mu_1,\dots,\mu_n\} \triangleright Q \xrightarrow{\alpha} Q'}$$

Definition 3.2 (*refusal transitional semantics*) The inference rules defining the transition relation $\xrightarrow{X}_r \subseteq \tilde{\mathbb{S}} \times \tilde{\mathbb{S}}$ where $X \subseteq \mathbb{A}$ are those in Table 3 where we replace the rules $\textsc{Read}_{t1}$ and $\textsc{Read}_{t2}$ with:

$$\textsc{Read}_t\ \frac{Q \xrightarrow{X}_r Q',\ \mathscr{U}(\{\mu_1,\dots,\mu_n\}) \cap (X \cup \{\tau\}) = \emptyset}{\{\mu_1,\dots,\mu_n\} \triangleright Q \xrightarrow{X}_r \underline{\{\mu_1,\dots,\mu_n\}} \triangleright Q'}$$

where $\mathscr{U}(\{\mu_1,\dots,\mu_n\}) = \{\alpha \mid \mu_i = \underline{\alpha} \text{ for some } i \in [1,n]\}$ and $\underline{\{\mu_1,\dots,\mu_n\}}$ is the set obtained from $\{\mu_1,\dots,\mu_n\}$ by replacing each α by $\underline{\alpha}$.

A term $Q \in \tilde{\mathbb{S}}$ is *read-guarded* if every subterm of Q of the form $\{\mu_1,\dots,\mu_n\} \triangleright Q'$ is in the scope of some action prefix $\mu.()$. $Q \in \tilde{\mathbb{S}}$ is *read-proper* if each subterm $Q_1 + Q_2$ is read-guarded and, for each subterm $\{\mu_1,\dots,\mu_n\} \triangleright Q_1$, Q_1 is read-guarded. We say that Q is *x-proper* if any free x is guarded in any subterm $Q_1 + Q_2$, $\{\mu_1,\cdots,\mu_n\} \triangleright Q_1$ and $\mathsf{rec}\, y.Q_1$. Q is *rec-proper* if for any subterm $\mathsf{rec}\, x.Q_1$, Q_1 is either read-guarded or x-proper. A term Q is *proper* if it is read- and rec-proper. Below, we will prove that proper terms have a reasonable semantics by relating them to PAFAS$_r$ processes with the same behaviour. An important feature of properness is that processes without read-prefixes are proper.

According to the definitions given so far, neither $P \equiv \{a\} \triangleright \{b\} \triangleright Q$ nor $P' \equiv \{a\} \triangleright Q' + \{b\} \triangleright Q$ are read-proper because of $\{b\} \triangleright Q$. An essential idea of reading is that it does not change the state of a process and therefore does not block other actions. With this, we should have $P \xrightarrow{b} P$, but really we have $P \xrightarrow{b} \{b\} \triangleright Q$. Similarly, we have $P' \xrightarrow{b} \{b\} \triangleright Q$ instead of $P' \xrightarrow{b} P'$. Hence, we exclude such processes. There is also a problem with the term $P \equiv \mathsf{rec}\, x.\{a\} \triangleright b.(c+x)$. Indeed, P can perform a b and evolve to $c + \mathsf{rec}\, x.\{a\} \triangleright b.(c+x)$ which is not read-proper. Since the body of this recursion is not read-guarded, x has to be treated as a read-prefix term, i.e. the body has to be x-proper. A subtle detail is the consideration of recursive subterms in the definition of x-proper. Without this detail, $Q \equiv \mathsf{rec}\, x.\{a\} \triangleright b.\mathsf{rec}\, y.(c.(c+y) \parallel_\emptyset x)$ would be proper. But, $Q \xrightarrow{b} \mathsf{rec}\, y.(c.(c+y) \parallel_\emptyset Q) \xrightarrow{c} (c + \mathsf{rec}\, y.(c.(c+y) \parallel_\emptyset Q)) \parallel_\emptyset Q$. Notice that $\mathsf{rec}\, y.(c.(c+y) \parallel_\emptyset Q)$, and hence $c + \mathsf{rec}\, y.(c.(c+y) \parallel_\emptyset Q)$, is not read-proper.

In contrast to the restriction to proper terms, we can freely use read-prefixes in PAFAS$_r$, see e.g. the process in Example 2.4; this would have the *wrong semantics* in PAFAS$_s$, i.e. if we change $r_{ij} \triangleright$ and $r^k_j \triangleright$ (for $i,j \in \{t,f\}$ and $k \in \{1,2\}$) into $\{r_{ij}\}\triangleright$ and $\{r^k_j\}\triangleright$. The restriction only makes sense because of Prop. 3.3, which requires a careful, detailed proof.

Proposition 3.3 *Let $Q \in \tilde{\mathbb{S}}$ be proper. $Q \xrightarrow{\alpha} Q'$ or $Q \xrightarrow{X}_r Q'$ implies Q' proper.*

Actually, the result in [10] is not correct since we used an insufficient restriction there. But, luckily the PAFAS$_s$ process we used to model Dekker's MUTEX algorithm is proper. This can been easily seen since proper processes are closed w.r.t. parallel composition and relabelling.

[5] To be formally precise: we have to replace all arrows $\mapsto$ in Table 1 by $\rightarrow$.

4 Expressivity of PAFAS$_s$

In this section we compare the expressivity of PAFAS$_s$ with that of PAFAS$_r$ and Petri nets. A first result shows that for each proper $Q \in \tilde{\mathbb{S}}$ there is a term in $\tilde{\mathbb{P}}$ whose behaviour is (timed) bisimilar and even isomorphic to that of Q.

Definition 4.1 (*timed bisimulation*) A binary relation $\mathscr{S} \subseteq \mathbb{P} \times \mathbb{P}$ over processes is a *timed bisimulation* if $(Q,R) \in \mathscr{S}$ implies, for all $\alpha \in \mathbb{A}_\tau$ and all $X \subset \mathbb{A}$:

- whenever $Q \overset{\alpha}{\rightsquigarrow} Q'$ ($Q \overset{\alpha}{\mapsto} Q'$, $Q \xrightarrow{X}_r Q'$) then, for some R', $R \overset{\alpha}{\rightsquigarrow} R'$ ($R \overset{\alpha}{\mapsto} R'$, $R \xrightarrow{X}_r R'$, resp.) and $(Q',R') \in \mathscr{S}$;
- whenever $R \overset{\alpha}{\rightsquigarrow} R'$ ($R \overset{\alpha}{\mapsto} R'$, $R \xrightarrow{X}_r R'$) then, for some Q', $Q \overset{\alpha}{\rightsquigarrow} Q'$ ($Q \overset{\alpha}{\mapsto} Q'$, $Q \xrightarrow{X}_r Q'$, resp.) and $(Q',R') \in \mathscr{S}$.

Two processes $Q,R \in \tilde{\mathbb{P}}$ are timed bisimilar (*bisimilar* for short, written $Q \sim R$) if $(Q,R) \in \mathscr{S}$ for some timed bisimulation $\mathscr{S}$. This definition is extended to open terms as usual; two open terms are bisimilar if they are so for all closed substitutions. It can be proved in a standard fashion that timed bisimilarity is a *congruence* w.r.t. all PAFAS$_r$ operators. The same definition, but omitting the reading transitions, applies to PAFAS$_s$.

We start by providing a translation function $[\![\_]\!]_r$ that maps terms in $\tilde{\mathbb{S}}$ into corresponding terms in $\tilde{\mathbb{P}}$; to regard $[\![\_]\!]_r$ as a function in the read-case, we have to assume that actions are totally ordered, and that the actions of a read-set are listed according to this order.

Definition 4.2 (*a translation function*) For $Q \in \tilde{\mathbb{S}}$ proper, $[\![Q]\!]_r$ is defined by induction on Q (subterms of Q are also proper) as follows :

Nil, Var, Pref :	$[\![\mathsf{nil}]\!]_r \equiv \mathsf{nil}$, $\quad [\![x]\!]_r \equiv x$, $\quad [\![\mu.P]\!]_r \equiv \mu.[\![P]\!]_r$
Read:	$[\![\{\mu_1,\ldots,\mu_n\} \triangleright Q]\!]_r \equiv \mu_1 \triangleright \ldots \triangleright \mu_n \triangleright [\![Q]\!]_r$
Sum:	$[\![Q_1 + Q_2]\!]_r \equiv [\![Q_1]\!]_r + [\![Q_2]\!]_r$
Par:	$[\![Q_1 \parallel_A Q_2]\!]_r \equiv [\![Q_1]\!]_r \parallel_A [\![Q_2]\!]_r$
Rel:	$[\![Q[\Phi]]\!]_r \equiv [\![Q]\!]_r[\Phi]$
Rec:	$[\![\mathsf{rec}\, x.Q]\!]_r \equiv \mathsf{rec}\, x.[\![Q]\!]_r$

This translation is pretty obvious, but its correctness is not; observe that Theorem 4.3 does not hold for all PAFAS$_s$ processes; cf. the processes $P \equiv \{a\} \triangleright \{b\} \triangleright Q$ and $P' \equiv \{a\} \triangleright Q' + \{b\} \triangleright Q$ at the end of Section 3. Function $[\![]\!]_r$ is injective on proper terms; except for the read-case, this is easy since $[\![]\!]_r$ preserves all other operators. In the read-case, Q is read-guarded, i.e. the top-operator of Q and $[\![Q]\!]_r$ is not $\triangleright$; the read-set can be read off from $[\![\{\mu_1,\ldots,\mu_n\} \triangleright Q]\!]_r$ as the maximal sequence of $\triangleright$-prefixes the term starts with. With this observation, the following result, together with Prop. 3.3, shows that $[\![]\!]_r$ is an isomorphism between labelled transition systems, if we restrict them, on the one hand, to proper terms and their transitions and, on the other, to the images of proper terms and the transitions of these images.

Theorem 4.3 For all proper $Q \in \tilde{\mathbb{S}}$:

1. $Q \xrightarrow{\alpha} Q'$ ($Q \xrightarrow{X}_r Q'$) implies $[\![Q]\!]_r \xrightarrow{\alpha} [\![Q']\!]_r$ ($[\![Q]\!]_r \xrightarrow{X}_r [\![Q']\!]_r$, resp.);
2. if $[\![Q]\!]_r \xrightarrow{\alpha} Q''$ ($[\![Q]\!]_r \xrightarrow{X}_r Q''$) then $Q \xrightarrow{\alpha} Q'$ ($Q \xrightarrow{X}_r Q'$) with $[\![Q']\!]_r \equiv Q''$.

The above theorem shows that the expressivity of proper PAFAS$_s$ processes is at most that of PAFAS$_r$. On the other hand, it is enough to model safe Petri nets with read-arcs. To illustrate the proof idea, which is based on a well-known view of a net as a parallel composition, consider an empty place of a net with preset $\{t_1,t_2\}$ and postset $\{t_3,t_4\}$, and being read by t_5 and t_6. This is translated into process P_0 with $P_0 \Leftarrow t_1.P_1 + t_2.P_1$ and $P_1 \Leftarrow \{t_5,t_6\} \triangleright (t_3.P_0 + t_4.P_0)$; P_1 models the marked place. All the analogous translations of places are composed in parallel, synchronising each time over all common actions (e.g. net transitions). Finally, a relabelling corresponding to the labelling of the net is applied.

Theorem 4.4 *For each safe Petri nets with read-arcs in [22] there is a bisimilar proper PAFAS$_s$ process.*

5 The backward translation from PAFAS$_r$ to PAFAS$_s$

In this section we study the problem whether PAFAS$_r$ is more expressive than PAFAS$_s$ or whether each PAFAS$_r$ term can be translated into a bisimilar proper PAFAS$_s$ term. We first exhibit a subset of $\tilde{\mathbb{P}}$ that is essentially the image of $[\![.]\!]_r$ and so has an easy translation; we say these terms are in *read normal form (RNF)* (see Def. 5.1). We then discuss how PAFAS$_r$ terms can be brought into RNF and illustrate, by means of examples, the problems of such a normalisation.

Definition 5.1 (*read normal form*) For PAFAS$_r$ terms, we define read-guarded, and x- and rec-proper as above except for considering read-action prefixes instead of read-set prefixes. We call such a term *ra-proper* if each subterm $Q_1 + Q_2$ is read-guarded, and for each subterm $\mu \triangleright Q'$ either Q' is read-guarded or $Q' \equiv \nu \triangleright Q''$. A term is *RNF* if it is rec- and ra-proper. The sets of terms and processes in RNF are denoted by $\tilde{\mathbb{P}}_{rn}$ and $\mathbb{P}_{rn}$, resp.

Below we provide the function that translates each $Q \in \tilde{\mathbb{P}}_{rn}$ into a proper term in $\tilde{\mathbb{S}}$. We will need an additional function to deal with read prefixes. A term such as $\mu_1 \triangleright Q$ is in RNF if either Q is read-guarded or, by iterative applications of Def. 5.1, Q has the form $\mu_1 \triangleright \cdots \triangleright \mu_n \triangleright Q_n$ where $Q_n \in \tilde{\mathbb{P}}_{rn}$ is read-guarded. In the latter case, the actions $\mu_1, \cdots, \mu_n$ must be collected in a read set. Since read sets cannot contain multiple copies (lazy and urgent) of the same action α, we use the following notation: if $\mu_1, \cdots, \mu_n$ are generic actions in $\mathbb{A}_\tau \cup \underline{\mathbb{A}}_\tau$, $[\![\mu_1, \cdots, \mu_n]\!]$ denotes the set of actions $\{\nu_1, \cdots, \nu_m\}$ such that: $\exists\, i \in [1,m]$ with $\nu_i = \underline{\alpha}$ iff $\exists\, j \in [1,n]$ with $\mu_j = \underline{\alpha}$; (2) $\exists\, i \in [1,m]$ with $\nu_i = \alpha$ iff $\exists\, j \in [1,n]$ such that $\mu_j = \alpha$ and, for each $k \in [1,n]$, $\mu_k \neq \underline{\alpha}$.

Definition 5.2 (*a translation function from $\tilde{\mathbb{P}}_{rn}$ to $\tilde{\mathbb{S}}$*) For $Q \in \tilde{\mathbb{P}}_{rn}$, we define the process term $[\![Q]\!]_s \in \tilde{\mathbb{S}}$ by induction on Q as in Definition 4.2 except for:

Read: $[\![\mu_1 \triangleright Q]\!]_s \equiv [\![\mu_1, \cdots, \mu_n]\!] \triangleright [\![Q_n]\!]_s$
if $Q \equiv \mu_1 \triangleright \cdots \triangleright \mu_n \triangleright Q_n$ and Q_n is read-guarded

With the laws L1 and L2 below, we can rearrange successive read-action prefixes in a process in RNF such that the result is in the image of $[\![]\!]_r$, which essentially proves the second item of following result.

Theorem 5.3 For all $Q \in \tilde{\mathbb{P}}_{rn}$:

1. $Q \xrightarrow{\alpha} Q'$ or $Q \xrightarrow{X}_r Q'$ imply $Q' \in \tilde{\mathbb{P}}_{rn}$;

2. Q and $[\![Q]\!]_s$ are timed bisimilar (in the sense of PAFAS$_s$).

Translating terms into read normal form

For translating a term that is *not* in read normal form, one idea is to use laws to rewrite the term into a bisimilar one in RNF. E.g. although $(a \triangleright b) + c$ does not belong to $\tilde{\mathbb{P}}_{rn}$, it has the same timed behaviour as $a \triangleright (b + c) \in \tilde{\mathbb{P}}_{rn}$, cf. L3.

Besides commutativity and associativity of $+$ and $\|$, we have shown the laws in Fig. 1. Here, $\Phi\{a \to \alpha\}$ denotes the relabelling function that renames a to α, and all other actions as Φ. For the discussion, we also write $[a \to \alpha]$ as a shorthand for $\Phi_I\{a \to \alpha\}$ where Φ_I is the identity relabelling function. The

L1	$\mu \triangleright (\nu \triangleright Q) \sim \nu \triangleright (\mu \triangleright Q)$
L2	$\alpha \triangleright (\mu \triangleright Q) \sim \mu \triangleright Q$, $\underline{\alpha} \triangleright (\mu \triangleright Q) \sim \underline{\alpha} \triangleright Q$ provided that $\mu \in \{\alpha, \underline{\alpha}\}$
L3	$(\mu \triangleright Q) + R \sim \mu \triangleright (Q + R)$
L4	$a \triangleright (Q_1 \|_A Q_2) \sim ((a \triangleright Q_1) \|_{A \cup \{a\}} (a \triangleright Q_2))$,
	$\underline{a} \triangleright (Q_1 \|_A Q_2) \sim ((\underline{a} \triangleright Q_1) \|_{A \cup \{a\}} (\underline{a} \triangleright Q_2))$ provided that $a \notin \mathscr{L}(Q)$
L5	$(\alpha \triangleright Q)[\Phi] \sim \Phi(\alpha) \triangleright (Q[\Phi])$, $(\underline{\alpha} \triangleright Q)[\Phi] \sim \underline{\Phi(\alpha)} \triangleright (Q[\Phi])$
L6	$(Q[\Phi])[\Psi] \sim Q[\Psi \circ \Phi]$
L7	$\mathsf{rec}\, x.Q \sim Q\{\mathsf{rec}\, x.Q/x\}$

Figure 1: A set of laws

idea of the translation into RNF is to perform rewriting by induction on the term size; since action-prefix, parallel composition and relabelling preserve RNF, these operators are no problem. Read-prefixes $\mu \triangleright Q$ can be dealt with distributing μ among Q's components. But choice and recursion pose still unsolved problems.

Regarding read prefixes, we have to show the stronger claim that for each Q in RNF we can normalise $\mu \triangleright Q$ in such a way that, for any variable y, y guarded in Q implies y guarded in the RNF, and if additionally Q is y-proper this is also preserved. The proof is by induction on Q; some cases are easy because $\mu \triangleright Q$ is in RNF itself (by the definition of RNF or by induction). We consider one of the three remaining cases, namely the Par-case. The Rel-case is easier, while the Rec-case is much more complicated. Their proofs can be found in the appendix. For a fresh action a we have:

$$\alpha \triangleright (Q_1 \|_A Q_2) \sim (a \triangleright (Q_1 \|_A Q_2))[a \to \alpha] \sim ((a \triangleright Q_1) \|_{A \cup \{a\}} (a \triangleright Q_2))[a \to \alpha]$$

by L4, and then we are done by induction. The case of an $\underline{\alpha}$-read-prefix is similar.

The case of choice is particularly tricky whenever one of the two alternatives is a parallel composition. Hence, we now concentrate on the following problem: *let $Q, R \equiv R_1 \|_A R_2$ be terms in RNF; is there an S in RNF such that $S \sim Q + R$?*

First, observe that we can rewrite Q into Q' by replacing all actions (also in relabellings) by fresh copies, such that Q' and R have disjoint sorts. Then, we can try to bring $Q' + R$ into RNF and finally apply a relabelling that 'undoes' the rewrite (cf. the last example above). This would give us a bisimilar term in RNF for $Q + R$. From now on we assume that Q and R have disjoint sorts.

If Q is *deterministic* (i.e. it never performs τ and never performs an action in two different ways), we have the law $Q + (R_1 \|_A R_2) \sim (Q + R_1) \|_{A \cup \mathscr{L}(Q)} (Q + R_2)$. Thus, to find S we now simply normalise the two components inductively. In general, this law fails: for $Q \equiv a.b + a.c$, $Q + R$ evolves with a into either b or c. But $(Q + R_1) \|_{A \cup \{a,b,c\}} (Q + R_2)$ can perform a and evolve into the deadlocked $b \|_{A \cup \{a,b,c\}} c$. A new idea that will work in many cases is to replace the second copy of Q by its 'top-part' that can perform the same time steps and the same initial actions as Q, but deadlocks after an ordinary action;

additionally, not all of $\mathscr{L}(Q)$ but only the initial actions are added to the synchronisation set: in our example, $((a.b+a.c)+R_1)\;\|_{A\cup\{a\}}\;(a+R_2)$ is bisimilar to $Q+R$ and could, in principle, be normalised inductively. This idea must be adapted in case of read prefixes. Consider $Q\equiv a\triangleright b.c$; here, the top-part is $a\triangleright b$, i.e. $Q+R$ is bisimilar to $(a\triangleright b.c+R_1)\;\|_{A\cup\{a,b\}}\;(a\triangleright b+R_2)$ (in particular, both terms remain unchanged when performing a). Another problem is that initial actions may also be performed later, e.g. in $Q\equiv a\triangleright b.a$; again, rewriting plus later relabelling helps. In the example, $Q+R$ is bisimilar to $((e\triangleright b.c+R_1)\;\|_{A\cup\{e,b\}}\;(e\triangleright b+R_2))[e\to a]$, and the terms $e\triangleright b.c+R_1$ and $e\triangleright b+R_2$ are again smaller than $Q+R$.

But what is the top-part for $Q\equiv a\;\|_{\emptyset}\;b$? Action a can be performed initially, but also after b. If we could transform Q into $a.b+b.a$, the top-part would be $a+b$, and using rewriting plus later relabelling solves the problem. But unfortunately $Q\sim a.b+b.a$ is wrong: when performing $1a$, these terms end up in $\mathsf{nil}\;\|_{\emptyset}\;\underline{b}$ and b resp., which are not timed bisimilar due to partial time step $\{b\}$.

Finding the top-part of parallel compositions seems to be related to finding a suitable expansion law. But even for standard PAFAS, such a law is not known. Thus, our general proof idea does not work so far, due to problems with choice terms. Also the treatment of recursion is not clear yet; an expansion law would certainly help. At least, we have identified a fragment of PAFAS$_r$ which does not have additional expressivity.

Theorem 5.4 *If all choice and recursive subterms of a PAFAS$_r$ process are in RNF then there is a bisimilar PAFAS$_s$ process.*

6 Conclusions and Future Work

We have studied two different ways to enhance PAFAS with non-blocking reading actions. We have first added reading in the form of a read-action prefix operator and proved that this adds expressivity w.r.t. fair behaviour. This operator is very flexible, but has a slightly complex semantics. To reduce complexity, we have introduced a read-set prefix operator with a simpler semantics, but with syntactic restrictions. For the second operator, it is not immediately clear whether its operational semantics models reading behaviour adequately. We could prove this by translating proper PAFAS$_s$ terms into PAFAS$_r$ terms with the same timed behaviour. We also show that PAFAS$_s$ is strong enough to model Petri nets with read-arcs.

It is still not clear whether PAFAS$_r$ is more expressive than the restricted PAFAS$_s$. We presented some ideas how a respective translation could work; these are based on some algebraic laws that are also interesting in their own right. In the future we will try to complete this translation. This is related to finding an expansion law for generic PAFAS$_r$ (and PAFAS) terms. Such an expansion law should also provide us with an axiomatisation for the full PAFAS language. Some results can be found in [21] where a fragment of the language that just consists of prefix and choice has been axiomatised.

We plan to use read prefixes for modelling systems and comparing their efficiency or proving them correct under the progress assumption. A first correctness proof (for Dekker's MUTEX algorithm) with the aid of the automated verification tool FASE has been presented in [9].

References

[1] S. D. Brookes, C. A. R. Hoare, A. W. Roscoe. A Theory of Communicating Sequential Processes. *Journal of the ACM* **31**, pp. 560–599, 1984, doi:10.1145/828.833.

[2] P. Bouyer, S. Haddad, P.A. Reynier. Timed Petri Nets and Timed Automata: On the Discriminating Power of Zeno Sequences. *Information and Computation* **206(1)**, 2008, doi:10.1016/j.ic.2007.10.004.

[3] F. Buti, M. Callisto De Donato, F. Corradini, M.R. Di Berardini and W. Vogler. Automated Analysis of MUTEX Algorithms with `FASE`. Proc. of GandALF 2011, doi:`10.4204/EPTCS.54.4`.

[4] D. Cacciagrano, F. Corradini, J. Aranda and F.D. Valencia. Linearity, Persistence and Testing Semantics in the Asynchronous Pi-Calculus. Proc. of EXPRESS'07, ENTCS 194(2), pp. 59–84, doi:`10.1016/j.entcs.2007.11.006`.

[5] S. Christensen, N. D. Hansen. Coloured Petri nets extended with place capacities, test arcs, and inhibitor arcs. In Applications of Theory of Petri Nets, LNCS 691, pp. 186–205, 1993, doi:`10.1.1.32.7925`.

[6] R. Cleaveland, G. Lüttgen, V. Natarajan. Priority in process algebra. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, Handbook of Process Algebra, pages 711–765. Elsevier Science Publishers, 2001.

[7] F. Corradini, M.R. Di Berardini and W. Vogler. Fairness of Actions in System Computations. *Acta Informatica* **43**, pp. 73 130, 2006, doi:`10.1007/s00236-006-0011-2`.

[8] F. Corradini, M.R. Di Berardini, and W. Vogler. Checking a Mutex Algorithm in a Process Algebra with Fairness. Proc. of CONCUR '06, pp. 142–157, LNCS 4137, 2006, doi:`10.1007/11817949_10`.

[9] F. Corradini, M.R. Di Berardini and W. Vogler. Time and Fairness in a Process Algebra with Non-blocking Reading. Proc. of SOFSEM'09, LNCS 5404, pp. 193–204, doi:`10.1007/978-3-540-95891-8_20`.

[10] F. Corradini, M.R. Di Berardini and W. Vogler. Time and Fairness in a Process Algebra with Non-blocking Reading. TR available at `www.informatik.uni-augsburg.de/en/chairs/swt/ti/staff/walter/publications`

[11] F. Corradini, W. Vogler, and L. Jenner. Comparing the Worst-Case Efficiency of Asynchronous Systems with PAFAS. *Acta Informatica* **38**, pp. 735–792, 2002, doi:`10.1007/s00236-002-0094-3`.

[12] G. Costa, C. Stirling. Weak and Strong Fairness in CCS. *Information and Computation* **73**, pp. 207–244, 1987, doi:`10.1016/0890-5401(87)90013-7`.

[13] G. Costa, C. Stirling. A Fair Calculus of Communicating Systems. *Acta Informatica* **21**, pp. 417–441, 1984, doi:`10.1007/BF00271640`.

[14] F. Crazzolara, G. Winskel. Events in security protocols. Proc. of 8th ACM conference on Computer and Communication Security, CCS'01, pp. 96–105, 2001, doi:`10.1145/501983.501998`.

[15] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[16] R. Milner. *Communication and Concurrency*. International series in computer science, Prentice Hall International, 1989.

[17] U. Montanari, F. Rossi. Contextual net. *Acta Informatica* **32**, pp. 545–596, 1995, doi:`10.1007/s002360050026`.

[18] U. Montanari, F. Rossi. Contextual occurrence nets and concurrent constraints programming. Proc. of Graph Transformation in Computer Science, LNCS 776, pp. 280–295, 1994, doi:`10.1007/3-540-57787-4_18`.

[19] M. Raynal. *Algorithms for Mutual Exclusion*. North Oxford Academic, 1986.

[20] G. Ristori. Modelling Systems with Shared Resources via Petri Nets. PhD thesis, Department of Computer Science, University of Pisa, 1994.

[21] W. Vogler, L. Jenner. Axiomatizing a Fragment of PAFAS. Electronic Notes in Theoretical Computer Science, 39(3) pp. 306–321, 2000, doi:`10.1016/S1571-0661(05)01225-9`.

[22] W. Vogler. Efficiency of Asynchronous Systems, Read Arcs and the MUTEX-problem. *Theoretical Computer Science* **275(1–2)**, pp. 589–631, 2002, doi:`10.1016/S0304-3975(01)00300-0`.

[23] D.J. Walker. Automated Analysis of Mutual Exclusion algorithms using CCS. *Formal Aspects of Computing* **1**, pp. 273–292, 1989, doi:`10.1007/BF01887209`.