ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Information and Computation 184 (2003) 311–342

Information and Computation

www.elsevier.com/locate/ic

Faster asynchronous systems☆

Walter Vogler*

Institut für Informatik, Universität Augsburg, Universitätsstraße 14, Augsburg D-86135, Germany

Received 10 August 1999; revised 29 October 2002

Abstract

A testing scenario in the sense of De Nicola and Hennessy is developed to measure the worst-case efficiency of asynchronous systems. The resulting testing-preorder is characterized with a variant of refusal traces and shown to satisfy some properties that make it attractive as a faster-than relation. Finally, one implementation of a bounded buffer is shown to be strictly faster than two others – in contrast to a result obtained with a different approach by Arun-Kumar and Hennessy.

Keywords: Asynchronous systems; Testing theory; Worst-case performance; Petri nets

1. Introduction

In the testing approach of [10], reactive systems are compared by embedding them – with a parallel composition operator $\|$ – in arbitrary test environments; if some system N_1 performs successfully in such an environment whenever a second system N_2 does, then N_1 is called an implementation of the specification N_2. There are two variants to evaluate the success: N_1 *may* satisfy a test O, if some run of $N\|O$ reaches success, which is signalled by a special action; N_1 *must* satisfy O, if every run of $N\|O$ reaches success. In other words, may-testing concentrates on the best case, must-testing on the worst case.

The testing scenario of De Nicola/Hennessy uses a basic behaviour notion to determine whether the success action is performed; this notion shows that the systems under consideration – which may

☆This work was partially supported by the DFG and the ESPRIT Basic Research Working Group 6067 CALIBAN (CAusal calcuLI BAsed on Nets). An extended abstract of this paper has appeared in CONCUR 95, LNCS 962.

* Fax: +49-821-598-2175.
E-mail address: walter.vogler@informatik.uni-augsburg.de.

0890-5401/03/$ - see front matter © 2003 Elsevier Science (USA). All rights reserved.
doi:10.1016/S0890-5401(03)00065-8

themselves be parallel systems – are asynchronous: one component of a system may perform any number of actions before another component performs its next action; such asynchronous systems are certainly of practical importance.

The classical testing approach was mostly studied in the process algebra community; it only takes into account the system functionality, i.e., which actions are performed, and it ignores performance issues. For many years, this abstraction was standard in the process algebra community: performance was regarded as too complicated for the time being. (Of course this has changed in the meantime.) In this paper, we do want to study the efficiency of asynchronous systems, which we model as Petri nets, but at the same time, we try to keep things simple.

For our testing scenario, we need a simple basic behaviour notion that tells us whether and also when success occurs. Petri nets viewed as synchronous systems have a simple behaviour notion: maximal-step sequences. This simplicity stems from the fact that they are defined for nets without explicit timing information – in contrast to the usual timed Petri nets (early references are [18,23]) or timed process algebras (see, e.g. [11,19]). Since there is no explicit timing information, maximal-step sequences are based on the assumption that each action takes the same time, say 1.

Since maximal-step sequences view a Petri net as a synchronous system, they do not cover all the functionality that the Petri net as an asynchronous system has. We will develop a behaviour notion that is just as easy as maximal-step sequences and enables us to judge the performance of asynchronous systems, i.e., takes into account all their functional behaviour.

A simple idea for studying the efficiency of systems in the testing scenario is presented in [27]: we can add a time bound d to our tests and require that some or every run reaches success within time d. It is no problem to measure the duration of a run, if the parallel system $N \| O$ is synchronous, i.e., if all components perform their actions according to a common global time scale; this case is treated in [27] as strict testing. In asynchronous systems, the components work at indeterminate relative speeds – the usual view is that they may idle unnecessarily or that actions may take more time than necessary, which interferes with time measurements. Actually, may-testing is not problematic: if some run reaches success in time, then intuitively all components are well-behaved in this run, i.e. they only idle to wait for some favourable action and otherwise they comply with global time, see [27]. Must-testing, on the other hand, is concerned with the more interesting worst-case behaviour, which would be to idle until time d is up; thus, no test at all must be satisfied, i.e., all systems are equally bad.

Nevertheless, we will develop a scenario of must-testing with time bounds for asynchronous systems and base a useful faster-than relation on it. For this, we take a different view at asynchronous systems: we will assume that there is an upper time bound for an action to occur; as explained above for maximal-step sequences, 1 is a natural choice for this bound in order to keep things simple. As a consequence of this view, a component is not any more allowed to idle or to take a lot of time with its current action; instead, since there is no positive lower time bound, we allow all others to work very fast in comparison such that the components still work at indeterminate relative speeds. This way we get a theory of asynchronous systems where there is an upper time bound for their performance. The idea (with different formalizations) goes back to at least [15,22]; also [17] treats asynchronous systems as having upper (and no lower) time bounds.

In this paper, we use (labelled, safe) Petri nets to model concurrent systems. Synchronous systems correspond to nets with the maximal-step firing rule (i.e., the components of the system working in lockstep are the transitions of the net), whereas asynchronous systems usually correspond to nets with

ordinary firing sequences. These Petri net notions together with parallel composition are defined in Section 2. The new firing rule for asynchronous systems with upper bound on action duration is introduced in Section 3. Note that we assume that actions have durations; no time is spent between actions (unless a component waits for a synchronization), i.e., we assume maximal progress. For simplicity, we use discrete time; since each transition takes at most 1, the comparatively fast actions do not take any time at all. As a consequence, we do not have to perform complicated calculations of real time, the state space of a finite system is finite, and in fact our asynchronous firing rule is a simple combination of the ordinary and the maximal-step firing rule. Based on an early version of the present paper, it has been shown that this approach coincides with an approach that follows the same idea, but uses continuous time [14].

It should be stressed that in asynchronous systems time cannot be used to guarantee correctness (e.g., by using time-outs); we only consider time as seen by an outside observer to measure the *efficiency* of these systems. This is done in Section 4, where our testing scenario is defined; a system N satisfies a test (O, d) if every run of $N \| O$ reaches success within d units of time, assuming that each action takes at most one unit. If N_1 satisfies all tests (O, d) which N_2 satisfies, then N_1 is an implementation of N_2. In particular, it might satisfy some (O, d), where N_2 only satisfies some (O, d'), $d < d'$, but not (O, d); hence, N_1 satisfies each test at least as fast or possibly faster than N_2, and we will call N_1 a *faster implementation*. (One reason to study asynchronous systems is that only for them one can expect to find an intuitive faster-than relation as argued in [19].) For comparison, we also have a look at testing based on maximal-step sequences as basic behaviour.

As usual in a testing approach, the definition of the implementation relation formalizes some intuition, but it is not easy to work with directly since it refers to all possible tests. Consequently, one needs a characterization that only refers to the systems that are compared. The main theorem is a characterization of the 'faster-implementation' relation with some timed variant of refusal traces; see [12,25,27] for similar timed variants. This characterization shows that, quite surprisingly, testing reveals quite strong information about the timed behaviour of the tested systems – although the tests themselves are asynchronous systems and have therefore 'weak temporal control'. The 'faster-implementation' relation turns out to be a precongruence for parallel composition. Using the characterization, some properties of the relation are shown in Section 5 which could serve as some kind of benchmarks for faster-than relations. In Section 6, we compare three implementations of a bounded buffer with our faster-than relation (using a simulation technique in the proof); one implementation PIPE is the usual concatenation of 1-buffers, the other two use an array in a circular fashion to store the content of the queue. Quite surprisingly, our theory reveals that under some circumstances one of the array implementations is slower than PIPE; in contrast to the approach of [1], it turns out that only one of the implementations is faster than the others, which are incomparable. Finally, related literature is discussed in the conclusion.

2. Petri nets and parallel composition

In this section, a very brief introduction to Petri nets is given. For further information the reader is referred to, e.g. [21,24]. We will deal with safe Petri nets (place/transition- or S/T-nets) whose transitions are labelled with actions from some infinite alphabet Σ or with the empty word λ. These actions are left uninterpreted; the labelling only indicates that two transitions with the same label from Σ represent

the same action occurring in different internal situations, while λ-labelled transitions represent internal, unobservable actions. Σ contains a special action ω, which we will need in our tests to indicate success.

Thus, a *labelled Petri net* $N = (S, T, W, l, M_N)$ (or just a *net* for short) consists of finite disjoint sets S of *places* and T of *transitions*, the *weight function* $W : S \times T \cup T \times S \to \{0, 1\}$, the *labelling* $l : T \to \Sigma \cup \{\lambda\}$, and the *initial marking* $M_N : S \to \{0, 1\}$. When we introduce a net N, then we assume that implicitly this introduces its components $S, T, W, \ldots$ If $W(x, y) = 1$, then (x, y) is called an *arc*; for each $x \in S \cup T$, the *preset* of x is ${}^\bullet x = \{y \mid W(y, x) = 1\}$ and the *postset* of x is $x^\bullet = \{y \mid W(x, y) = 1\}$.

- A *multiset* over a set X is a function $\mu : X \to \mathbb{N}_0$. We identify each $x \in X$ with the multiset that is 1 for x and 0 everywhere else; a subset Y of X is identified with the multiset that is 1 for $y \in Y$ and 0 everywhere else; we only make light use of these identifications. For multisets, multiplication with scalars from $\mathbb{N}_0$ and addition is defined elementwise.
- A *marking* is a multiset over S, a *step* is a multiset over T. A step μ is *enabled* under a marking M, denoted by $M[\mu\rangle$, if $\sum_{t \in \mu} \mu(t) \cdot {}^\bullet t \leqslant M$. One usually says that the step is enabled under the *maximal-firing* rule, if additionally: whenever $M[\mu'\rangle$ and $\mu \leqslant \mu'$ (transition-wise), then $\mu = \mu'$.
  If $M[\mu\rangle$ and $M' = M + \sum_{t \in \mu} \mu(t) \cdot t^\bullet - \sum_{t \in \mu} \mu(t) \cdot {}^\bullet t$, then we denote this by $M[\mu\rangle M'$ and say that μ can *occur* or *fire* under M yielding the *follower marking* M'. We also say that the transitions of μ can fire *concurrently*. Since transitions are special steps, this also defines $M[t\rangle$ and $M[t\rangle M'$ for $t \in T$.
- This definition of enabling and occurrence can inductively be extended to sequences as usual: a sequence w of steps is *enabled* under a marking M, denoted by $M[w\rangle$, and yields the follower marking M' when *occurring*, denoted by $M[w\rangle M'$, if $w = \lambda$ and $M = M'$ or $w = w'\mu$, $M[w'\rangle M''$ and $M''[\mu\rangle M'$ for some marking M''. If w is enabled under the initial marking, then it is called a *step sequence*, or – in case that $w \in T^*$ – a *firing sequence*. If all the steps are enabled under the maximal-firing rule, then one usually calls w a *maximal-step sequence*.

We can extend the labelling of a net to steps by $l(\mu) = \sum_{t \in T, l(t) \neq \lambda} \mu(t) \cdot l(t)$, where the treatment of the empty sum is slightly subtle: if we deal with steps in general, we define the empty sum to be the empty word; hence, $l(\mu)$ is a nonempty multiset of visible actions or λ – just as $l(t)$ for a transition t is a visible action or λ – such that internal transitions are completely unobservable. If we deal with maximal steps as in the definition of a maximal-step trace in the next itemized paragraph below, then we define the empty sum to be the empty set: such a step corresponds to the passing of one time unit, and if nothing visible happens, we at least want to observe the passage of time.

Then we can extend the labelling also to sequences of steps, transitions and maximal steps as usual, i.e., homomorphically. Next, we lift the enabledness and firing definitions to the level of actions:

- A sequence v of multisets over Σ is *enabled* under a marking M, denoted by $M[v\rangle\rangle$, if there is some w with $M[w\rangle$ and $l(w) = v$. If $M = M_N$, then v is called a *step trace*; similarly, a maximal-step sequence w gives rise to a *maximal-step trace* v – recall how internal transitions are treated in this case; if $w \in T^*$, then v is called a *trace*. We call two nets *step equivalent* if they have the same step traces. The *language* $L(N)$ is the set of all traces of N, and *language equivalence* and *language inclusion* are defined accordingly.
- For a marking M the set $[M\rangle$ of markings *reachable* from M is defined as $\{M' \mid \exists w \in T^* : M[w\rangle M'\}$. A marking is called *reachable* if it is reachable from M_N. The net is *safe* if $M(s) \leqslant 1$ for all places s and reachable markings M.

- Two not necessarily distinct transitions t_1 and t_2 are *concurrently* enabled under some marking M if $M[t_1 + t_2\rangle$. A transition t is *self-concurrent*, if $M[2t\rangle$ for some reachable marking M. An action $a \in \Sigma$ is *autoconcurrent*, if $M[2a\rangle\rangle$ for some reachable marking M.

General assumption. All nets considered in this paper are safe and without isolated transitions, i.e., ${}^\bullet t \cup t^\bullet \neq \emptyset$ for all transitions t.

This implies that all nets in this paper are free of self-concurrency, but it does not exclude autoconcurrency.

For each set A of transitions or actions, A^+ denotes a disjoint copy of A whose elements are denoted a^+, $a \in A$. We call $t^+ \in T^+$ or $a^+ \in \Sigma^+$ the *start* of t or a, while t and a themselves are called *complete*. The idea is that t^+ or a^+ only empties the corresponding preset. Note that we use A^* to denote – as usual – the set of all sequences over A.

In the following, we will use a reformulation of the maximal-firing rule: In a maximal step as defined above, we have a collection of transitions that start together, then they are performed together, and finally they end together. In our reformulation, we will write down a sequence of the corresponding transition starts – each emptying the respective preset – followed by a σ denoting the end of the sequence and the time unit during which the transitions are performed. The σ also implicitly denotes the end of the transitions that have started before. This reformulation stresses the assumption we will make throughout that time is spent during actions, not between actions as, e.g., in Timed CSP [11], where actions are instantaneous.

The price that we have to pay is that the intermediate states are not anymore described by a marking alone, but also by a set of transitions that have already started but have not finished yet. The advantage is a linear notation and a rather local firing rule; the main advantage – related to this locality – will become clear when we define simulations in Section 5.

Definition 1. An *instantaneous description* (*ID*) of a net N is a pair (M, C) consisting of a marking M of N and a set $C \subseteq T$ of *current(ly firing) transitions*. The *initial ID* is $ID_N = (M_N, \emptyset)$. If we introduce *ID* or ID', etc., this implicitly introduces its components M and C or M' and C', etc.

For *ID*'s (M, C) and (M', C') and $\varepsilon \in T \dot\cup T^+ \dot\cup \{\sigma\}$ ($\dot\cup$ denotes disjoint union), we write $(M, C)[\varepsilon\rangle_m (M', C')$ if one of the following cases applies:

(i) $\varepsilon = t^+ \in T^+,\ M[t\rangle,\ M' = M - {}^\bullet t$ and $C' = C \cup \{t\}$,

(ii) $\varepsilon = \sigma,\ \neg M[t\rangle$ for all $t \in T,\ M' = M + \sum_{t \in C} t^\bullet$ and $C' = \emptyset$.

This maximal-firing rule is extended to sequences as usual. If $ID_N[w\rangle_m ID$, then w is a *maximal-step firing sequence* and *ID* is *maximal-reachable*. The set of maximal-step firing sequences is denoted by $MFS(N)$.

These notions are lifted to the action level as follows: if $ID[t^+\rangle_m ID'$, we write $ID[l_N(t)^+\rangle\rangle_m ID'$ for visible t and $ID[\rangle\rangle_m ID'$ otherwise; if $ID[\sigma\rangle ID'$, we write $ID[\sigma\rangle\rangle_m ID'$. Again, we extend this action firing rule to sequences; if $ID_N[w\rangle\rangle_m ID$, then w is called a *maximal-step trace* of N. The *maximal-step language* $ML(N)$ consists of the maximal-step traces of N. Nets are *maximal-step equivalent* if they have the same maximal-step language.

The σ's subdivide a maximal-step firing sequence or trace w into *rounds*, i.e., a round is the prefix of w up to the first σ or a substring of w reaching from one σ to the next; if w carries on after the last σ, then this suffix is (part of) the last round of w. If some t^+ or a^+ occurs in a round, we say that t or a *starts* in this round and *ends* in the next.

The following should be obvious: if $ID_N[w\rangle(M, C)$ and w' is obtained from w by deleting all $+$-signs and all σ's, then $M_N[w'\rangle(M + \sum_{t\in C} t^\bullet)$. Hence, since we are only interested in reachable ID's and our nets are safe, we can restrict attention to ID's (M, C) where M is actually a set, i.e., a function $S \to \{0, 1\}$. Furthermore, if (M, C) is a reachable ID and $(M, C)[t^+\rangle(M', C')$, then $t \notin C$, i.e., in Clause (i) above, t is really added to the set of current transitions when it starts firing.

This should make it clear that Definition 1 is indeed only a reformulation of the corresponding definition given before in the following sense: if a sequence of multisets of actions is a maximal-step trace and we transform each multiset into a corresponding sequence of action starts followed by a σ, we get a maximal-step trace as in Definition 1 (which ends with a σ). If we take a maximal-step trace as defined in Definition 1 that ends with a σ and collect for each round the starting actions in a multiset, we get a sequence of multisets that is a maximal-step trace. Since each maximal-step trace as in Definition 1 can be extended to end with a σ, the maximal-step traces according to one definition can be determined from the maximal-step traces according to the other definition; thus, e.g., the respective equivalences coincide.

Finally, we introduce parallel composition $\|_A$ with synchronization inspired from TCSP. If we combine nets N_1 and N_2 with $\|_A$, then they run in parallel and have to synchronize on actions from A. To construct the composed net, we have to combine each a-labelled transition t_1 of N_1 with each a-labelled transition t_2 from N_2 if $a \in A$. Before giving the formal definition we show an example.

Fig. 1 shows in the upper half on the left the producer *Prod*, who repeatedly *p*roduces and *s*ends items, and on the right the channel component $Chan_2$, where *s*end- and *r*ecieve-actions can be performed. $Chan_2$ consists of two independent, isomorphic components *Chan*, hence it can be written as $Chan \|_\emptyset Chan$.

If we combine these nets with $\|_{\{s\}}$, the producer can send the items over the two channels; graphically, (a copy of) each s-transition of the first net is simply merged with each s-transition of the other net; the result is shown in the lower half of Fig. 1.

In the definition of parallel composition, $*$ is used as a dummy element, which is formally combined, e.g., with those transitions that do not have their label in the synchronization set A. (We assume that $*$ is not a transition or a place of any net.)

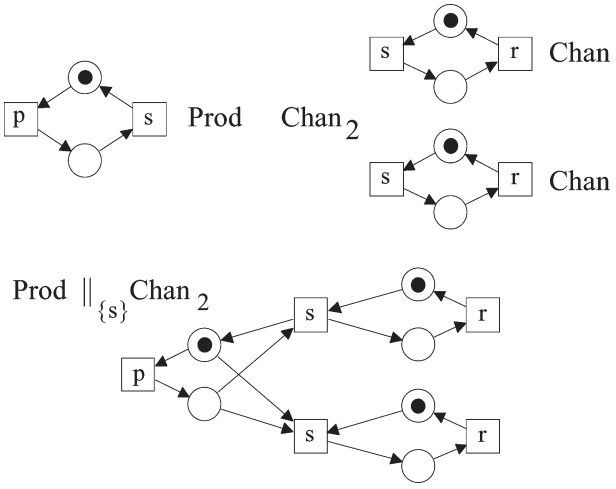

Fig. 1.

Let N_1, N_2 be nets, $A \subseteq \Sigma$. Then the *parallel composition* $N = N_1 \|_A N_2$ *with synchronization* over A is defined by

$$
\begin{aligned}
S &= S_1 \times \{*\} \cup \{*\} \times S_2 \\
T &= \{(t_1, t_2) \mid t_1 \in T_1, t_2 \in T_2, l_1(t_1) = l_2(t_2) \in A\} \\
&\quad \cup \{(t_1, *) \mid t_1 \in T_1, l_1(t_1) \notin A\} \\
&\quad \cup \{(*, t_2) \mid t_2 \in T_2, l_2(t_2) \notin A\}
\end{aligned}
$$

$$
W((s_1, s_2), (t_1, t_2)) = \begin{cases} W_1(s_1, t_1) & \text{if } s_1 \in S_1, t_1 \in T_1, \\ W_2(s_2, t_2) & \text{if } s_2 \in S_2, t_2 \in T_2, \\ 0 & \text{otherwise,} \end{cases}
$$

$$
W((t_1, t_2), (s_1, s_2)) = \begin{cases} W_1(t_1, s_1) & \text{if } s_1 \in S_1, t_1 \in T_1, \\ W_2(t_2, s_2) & \text{if } s_2 \in S_2, t_2 \in T_2, \\ 0 & \text{otherwise,} \end{cases}
$$

$$
l((t_1, t_2)) = \begin{cases} l_1(t_1) & \text{if } t_1 \in T_1, \\ l_2(t_2) & \text{if } t_2 \in T_2, \end{cases}
$$

$$
M_N = M_{N_1} \dot{\cup} M_{N_2}, \text{ i.e. } M_N((s_1, s_2)) = \begin{cases} M_{N_1}(s_1) & \text{if } s_1 \in S_1, \\ M_{N_2}(s_2) & \text{if } s_2 \in S_2. \end{cases}
$$

Parallel composition is an important operator for the modular construction of nets. In the present paper, the main purpose of this operator is to combine a net N with a test net. We use $\|$ to denote $\|_{\Sigma - \{\omega\}}$. Designing suitable test nets O and looking at the behaviour of $N \| O$, we can get information on the behaviour of N. The net O may also be regarded as an observer of N. For the general approach of testing, see [10].

3. Asynchronously timed behaviour

In this section, we define our new asynchronous behaviour and compare it shortly with existing firing rules. As explained in the introduction, we assume that the duration of an action is at most 1 and we use discrete time; furthermore, we assume maximal progress. Hence, in our new firing rule, every enabled transition will start firing immediately (unless it is disabled immediately) and will take up to one unit of time to finish, i.e., it will take time 1 or it might act faster and finish immediately. The former corresponds to the maximal-step firing rule as explained in Section 2, the latter to the ordinary firing rule; hence, our new firing rule will be a combination of both. Since we include the ordinary firing rule, we also consider behaviour that is impossible if transitions fire in lockstep, i.e., are synchronized via global time. Thus, in judging the efficiency later on, we will take into account *all* behaviour possible for an asynchronous system.

Jenner and Vogler [14] consider the same sort of behaviour, but with continuous time; there, the duration of a transition can be any real number between 0 and 1. This leads to a more complicated firing rule, since for the current actions one has to keep track of the remaining durations, and it leads to an infinite state space; it is shown that the resulting testing relation is nevertheless the same as the one we will consider here.

As in Definition 1, the asynchronous firing sequences we are going to define consist of rounds separated by σ's. A round is what happens at one instant: some transitions start firing and finish immediately and we list these transitions; other transitions just start and we list their starts. Since an enabled transition has to start immediately, time can only go on (signalled by σ) if no transition is enabled; thus, the transitions that have started but not finished in the round form a maximal step as in Definition 1. These transitions use up their time while the σ occurs, i.e. they end in the next round.

Thus, a round in fact begins with a sequence of transition ends, which we leave implicit – their starts are listed in the round before. We choose to list the fast transitions next (enforced by the condition $C = \emptyset$ in Part (i) below) and the transition starts afterwards. This design decision will be discussed below.

Definition 2. For *ID*'s (M, C) and (M', C') of a net N and $\varepsilon \in T \dot\cup T^+ \dot\cup \{\sigma\}$, we write $(M, C)[\varepsilon\rangle(M', C')$ if one of the following cases applies:

(i) $\varepsilon = t \in T,\ M[t\rangle M'$ and $C = C' = \emptyset$,

(ii) $\varepsilon = t^+ \in T^+,\ M[t\rangle,\ M' = M - {}^\bullet t$ and $C' = C \cup \{t\}$,

(iii) $\varepsilon = \sigma,\ \neg M[t\rangle$ for all $t \in T,\ M' = M + \sum_{t\in C} t^\bullet$ and $C' = \emptyset$.

This asynchronous firing rule is extended to sequences as usual. If $ID_N[w\rangle ID$, then w is an *asynchronous(ly timed) firing sequence* and *ID* is *reachable*. The set of asynchronous firing sequences is denoted by $AFS(N)$.

These notions are lifted to the action level as above; then, the action firing rule is extended to sequences. If $ID_N[w\rangle\!\rangle ID$, then w is called an *asynchronous(ly timed) trace* of N. The *asynchronous language* $AL(N)$ consists of the asynchronous traces of N. Nets are *asynchronously equivalent* if they have the same asynchronous language and *asynchronous inclusion* is defined similary.

The σ's subdivide an asynchronous firing sequence or trace w into *rounds*, i.e., a round is the prefix of w up to the first σ or a substring of w reaching from one σ to the next; if w carries on after the last σ, then this suffix is (part of) the last round of w.

In effect, an asynchronous firing sequence simply is a sequence of rounds, each consisting of an ordinary firing sequence followed by a maximal step and closed by a σ, where we use the linear notation of maximal steps as in Definition 1. We could also use the alternative, more usual multiset-notation explained before Definition 1 and define asynchronous firing sequences and asynchronous traces simply using markings instead of *ID*'s. This simplicity is one of the reasons for the design decision mentioned above. Though this simplicity is somewhat hidden due to the linear notation (which has consequences for the later use of simulations), we can think of the asynchronous language and the related notions in this simple fashion.

Our design decision to list fast transitions before the transition starts (i.e to require that C, and hence C', is empty in 2 (i)) also leads to a smoother presentation – i.e., to a closer relation to the asynchronous refusal traces we will define later.

The alternative to our decision would be to drop '$= \emptyset$' from 2 (i). As a consequence, we would have a mixed sequence of transition starts and complete (i.e., fast) transitions between two σ's; thus, we can observe a fast transition after the start and before the end of another transition, and this in fact looks like a more intuitive definition of an observation, and it gives more asynchronous firing sequences. This is a very important point: we will argue later that a certain kind of behaviour is observable using tests; but

the testing definition 4 in turn is based on some sort of basic observation; the latter is defined above in 2, and it is essential that it is intuitively convincing.

The justification of our '$= \emptyset$'-condition is that it does not make any difference whether we use Definition 2 or its alternative in the testing definition. Intuitively, the reason for this is that all things happening in a round happen at one instant anyway, so that their order is not really relevant. Technically speaking, the testing Definition 4 only takes into account asynchronous firing sequences of a certain duration not containing ω; if the alternative we are discussing at present would give such an asynchronous firing sequence that is not allowed according to 2, then we could reorder it such that it is allowed and serves the same purpose in 4. The point is that a transition start t_1^+ just removes tokens, hence a fast transition t_2 occurring afterwards can just as well occur before and the same *ID* is reached after $t_1^+ t_2$ and $t_2 t_1^+$; repeating this, we can reorder any relevant additional asynchronous firing sequence allowed in the alternative such that it fits 2, has the same duration and still does not contain ω.

Remark. Just as remarked after Definition 1, if (M, C) is an *ID* reached by an asynchronous firing sequence, then M is a safe marking and whenever we apply Definition 2 (ii), then $t \notin C$, i.e., t is added to the set C of current transitions when it starts firing.

Another worry with the above definition could be the following: if we use Clause (i) only, we get the usual firing sequences as a special case of our asynchronous firing sequences. These sequences take no time at all, and this is not very realistic; but it does not pose a problem either: ultimately, we will only be interested in worst-case behaviour, i.e., in sequences that take a lot of time. Hence, the presence of unrealistically fast asynchronous firing sequences does not influence our results regarding testing.

We thus see that $AL(N)$ is an extension of $L(N)$, and in fact a fairly conservative one: if we take an asynchronous trace in our notation and delete all $+$-signs and σ's, we get an ordinary trace; thus, asynchronous traces are just ordinary traces where $+$-signs and σ's are added to give some timing information.

On the other hand, if we never use Clause (i), we get the maximal-step sequences; in the absence of internal transitions, use of Clause (i) is always indicated on the action level by the occurrence of a complete action. Taking these together, we conclude:

Proposition 3.

(i) *Asynchronous equivalence implies language equivalence and, for nets without internal transitions, maximal-step equivalence.*

(ii) *Asynchronous inclusion implies language inclusion and, for nets without internal transitions, inclusion of maximal-step traces.*

The following examples demonstrate that several implications involving asynchronous equivalence do not hold in general. The first example in Fig. 2 consists of two asynchronously equivalent nets, where only the net on the left can perform a under the maximal-step firing rule; hence, asynchronous equivalence does not imply maximal-step equivalence in general. Of course, this implies that also asynchronous inclusion does not in general imply inclusion of maximal-step traces.

For the reverse implication, consider the maximal-step equivalent nets in Fig. 3; only the net on the right can perform ac under the asynchronous firing rule.

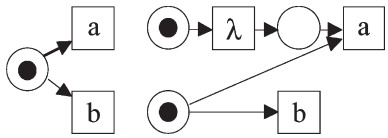

Fig. 2.

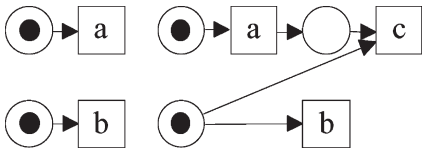

Fig. 3.

The nets of the third example (Fig. 4) are again asynchronously equivalent; to see this, observe that any asynchronous trace containing c or c^+ cannot contain a^+, b^+, d^+ or d, hence the c or c^+ is preceded by ab or ba. Only the net on the left can perform the step sequence $\binom{a}{b}c$, and therefore asynchronous equivalence does not imply step-sequence equivalence.

For the reverse implication, consider the step-sequence equivalent nets in Fig. 5; they are even process-equivalent, compare, e.g. [26, p.18]. But only the net on the right has the asynchronous trace $a^+\sigma$.

4. Asynchronously timed testing

In this section, we will define our testing scenario and prove a characterization for the resulting implementation preorder; for comparison, we will also look at a testing scenario based on synchronous behaviour as defined by maximal-step traces.

When testing a net, we apply parallel composition to embed it into a testing environment and consider the behaviour of the composed system. Before defining our testing scenario in detail let us look at an

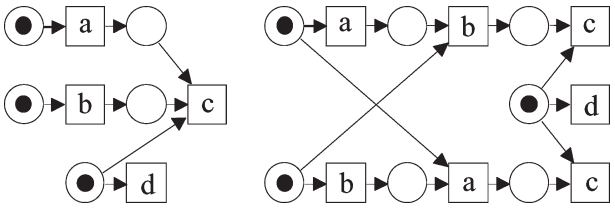

Fig. 4.

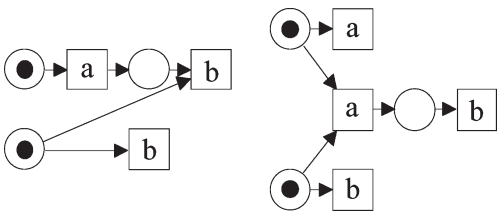

Fig. 5.

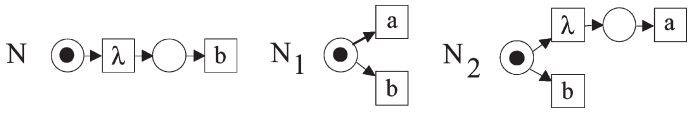

Fig. 6.

example, showing how the asynchronous firing rule behaves in combination with parallel composition.

Consider the nets N, N_1 and N_2 of Fig. 6. Although N_1 on its own does not allow σ under the asynchronous firing rule (due to maximal progress), $N\|N_1$ has the asynchronous trace σb; this shows that N_1 is willing to idle if the environment does not allow any of its activities. On the other hand, $\sigma b \notin AL(N\|N_2)$; if time passes and b is not possible, N_2 has to perform the internal action (maximal progress), thus deciding the choice to the disadvantage of b.

The same holds if we use the maximal-step firing rule; the difference is that under this rule $N\|N_2$ will definitely not perform b, while $b \in AL(N\|N_2)$.

We now come to the definition of testing. Recall that $\|$ requires synchronization of all actions except ω.

Definition 4. A net is *testable*, if none of its transitions is labelled with ω. An (*asynchronously*) *timed test* is a pair (O, d), where O is a net (the *test net*) and $d \in \mathbb{N}$ (the *time bound*). A testable net N *satisfies* a timed test (O, d), if each $w \in AL(N\|O)$ with d or more σ's contains some ω or ω^+. For testable nets N_1 and N_2, we call N_1 a *faster implementation* of N_2, $N_1 \sqsupseteq N_2$, if N_1 satisfies all timed tests that N_2 satisfies.

This version of testing is a form of must-testing: *every* run that lasts at least d units of time must be successful, indicated by ω or ω^+. The corresponding may-testing is nothing new, since it disregards the time bound: if some asynchronous trace of $N\|O$ contains ω or ω^+, then there is also such a trace without any σ's, which meets any time bound, and this trace corresponds to an ordinary trace. Hence, may-testing in our setting coincides with ordinary may-testing; we will not study it any further, apart from a remark at the end of this section.

It is usual to call N_1 an implementation of N_2, if N_1 satisfies all the tests N_2 satisfies – and possibly some more. Here, it might be the case that N_2 satisfies some $(O, d+1)$, but not (O, d), while N_1 satisfies (O, d) (– hence, obviously also $(O, d+1)$). Therefore, we call N_1 a *faster* implementation since, using it in any environment, success will never be reached slower, while in some environments it might even be reached faster; we will see below that, indeed, $\sqsupseteq$ satisfies some properties one might expect of a 'faster-than' relation.

For comparison, we also introduce testing based on maximal-steps.

Definition 5. A testable net N *m-satisfies* a timed test (O, d), if each $w \in ML(N\|O)$ with d or more σ's contains some ω^+. For testable nets N_1 and N_2, we call N_1 an *m-implementation* of N_2, if N_1 m-satisfies all timed tests that N_2 m-satisfies.

As already explained in the introduction, the implementation relation in a testing approach formalizes some intuitive notion, but it is not easy to work with directly since it involves all possible

tests. Our aim is now to characterize the faster-implementation relation in a way that refers only to the two nets under consideration. For this purpose we introduce the *ART*-semantics below. Its idea is to replace the maximal steps in an asynchronous trace by steps that are maximal with respect to internal actions and to *some* visible actions; no further instances of these actions are possible at the respective time, i.e., by listing these actions as a set X we get something like a refusal trace. The sets X replace the σ's in an asynchronous trace. (Observe the close similarity to Definition 2.) The same idea was behind the *SRT*-semantics in [27], which we generalize here to nets with internal actions.

Definition 6. For $\varepsilon \in T \dot{\cup} T^+ \dot{\cup} \dot{\mathfrak{P}}(\Sigma)$ and instantaneous descriptions (M, C) and (M', C') we write $(M, C)[\varepsilon\rangle_r (M', C')$ if one of the following cases applies:

(i) $\varepsilon = t \in T$, $M[t\rangle M'$ and $C = C' = \emptyset$,

(ii) $\varepsilon = t^+ \in T^+$, $M[t\rangle$, $M' = M - {}^\bullet t$ and $C' = C \cup \{t\}$,

(iii) $\varepsilon = X \subseteq \Sigma$, $\neg M[t\rangle$ for all $t \in T$ with $l(t) \in X \cup \{\lambda\}$, $M' = M + \sum_{t \in C} t^\bullet$ and $C' = \emptyset$.

The corresponding sequences are called *asynchronous refusal firing sequences*, their set is denoted by $ARFS(N)$. We lift this to the action level as above; in particular, we write $ID[X\rangle\!\rangle_r ID'$ if $ID[X\rangle_r ID'$ and call X a *refusal set*. The corresponding sequences are called *asynchronous refusal traces* with set $ART(N)$, and they induce *ART-equivalence*.

Again, an asynchronous refusal firing sequence or trace w is subdivided into *rounds*, this time by the refusal sets: a round is the prefix of w up to the first refusal set or a substring of w reaching from one refusal set to the next; if w carries on after the last refusal set, then this suffix is (part of) the last round of w. If $t^+ \in T^+$ or $a^+ \in \Sigma^+$ occurs in some round, we say that t or a *starts* in this round and *ends* in the next; if $t \in T$ or $a \in \Sigma$ occurs in some round, we say that t or a *starts* and *ends* in this round.

We write $ID[\varepsilon\rangle_{mr} ID'$, if we only use Clause (ii) and (iii) from above, and consequently define *m-refusal firing sequences* with set $MRFS(N)$ and *m-refusal traces* with set $MRT(N)$, inducing *MRT-equivalence*.

For a net N without internal transitions, $MRT(N)$ essentially coincides with $SRT(N)$ as defined in [27]. We note some useful properties, where the first two parts are analogues to results for ordinary failure semantics.

Proposition 7. *Let N be a net with $l(T) \cap \Sigma = A$; let $X \subseteq Y \subseteq \Sigma$ and uYv, $w \in ART(N)$.*

(i) $uXv \in ART(N)$,

(ii) $u\ (Y \cup (\Sigma - A))\ v \in ART(N)$,

(iii) $w\emptyset \in ART(N)$,

(iv) *If w' is obtained from w by permuting some sequences of action starts in w, then $w' \in ART(N)$.*

Proof. The first two parts are immediate from the definition; for the third part, one can extend w by starting internal transitions according to Definition 6 (ii) until no further internal transition is enabled under the remaining marking and then add $\emptyset$ according to Definition 6 (i). For the fourth part, recall that the sequences of action starts are just another notation for a step. □

We now prepare our characterization result; we show how the *ART*-semantics is related to the asynchronous language and that it is compatible with parallel composition.

Proposition 8.

(i) *For nets* N_1 *and* N_2, $ART(N_1) \subseteq ART(N_2)$ *implies* $AL(N_1) \subseteq AL(N_2)$ *and, analogously,* $MRT(N_1) \subseteq MRT(N_2)$ *implies* $ML(N_1) \subseteq ML(N_2)$.

(ii) *ART-equivalence implies asynchronous equivalence, MRT-equivalence implies maximum-step equivalence.*

(iii) *For nets without internal transitions, ART-equivalence implies MRT-equivalence, which implies step-equivalence.*

Proof.

(i) Consider only those traces in $ART(N_i)$ or $MRT(N_i)$ where all refusal sets equal Σ. Replacing Σ by σ gives $AL(N_i)$ or $ML(N_i)$.

(ii) by (i)

(iii) The first part corresponds to Proposition 3. For the second part consider only those m-refusal traces, where all refusal sets are empty; the actions between two $\emptyset$'s form a step (after removal of the $+$-signs). □

To see that *ART*-equivalence does not in general imply step-sequence equivalence, replace action d in Fig. 4 by λ. Fig. 7 shows two nets without internal transitions that are *ART*-equivalent, but not process-equivalent. (Again, see, e.g. [26, p.18] for this notion.)

Next we define the parallel composition of asynchronous and m-refusal traces in order to describe the behaviour of a composed net.

Definition 9. Let $u, v \in (\Sigma \cup \Sigma^+ \cup \mathfrak{P}(\Sigma))^*$, $A \subseteq \Sigma$. Then $u \|_A v$ is the set of all $w \in (\Sigma \cup \Sigma^+ \cup \mathfrak{P}(\Sigma))^*$ such that for some n $u = u_1 \cdots u_n$, $v = v_1 \cdots v_n$, $w = w_1 \cdots w_n$ and for $i = 1, \ldots, n$:

- $u_i = v_i = w_i \in A \cup A^+$,
- $u_i = w_i \in (\Sigma - A) \cup (\Sigma - A)^+$ and $v_i = \lambda$,
- $v_i = w_i \in (\Sigma - A) \cup (\Sigma - A)^+$ and $u_i = \lambda$,

or

- $u_i, v_i, w_i \in \mathfrak{P}(\Sigma)$ and $w_i \subseteq ((u_i \cup v_i) \cap A) \cup (u_i \cap v_i)$.

This definition works as usual: when composing u and v, actions from A are merged, while others are interleaved; in w, actions from A can be refused if they are refused in u or v , while the others can only be refused if they are refused in both, u and v. Now the next theorem shows that composition of traces corresponds to composition of nets.

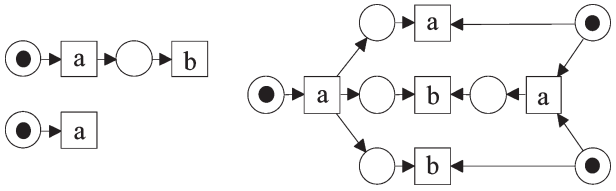

Fig. 7.

Theorem 10. *Let N_1, N_2 be nets and $A \subseteq \Sigma$. Then*

- $ART(N_1 \|_A N_2) = \bigcup\{u \|_A v \mid u \in ART(N_1),\ v \in ART(N_2)\}$,
- $MRT(N_1 \|_A N_2) = \bigcup\{u \|_A v \mid u \in MRT(N_1),\ v \in MRT(N_2)\}$.

Proof. The proof is similiar to, e.g., the proof of [26, Theorem 3.2.4] or of [27, Theorem 6.5]. One has to define that an *ID* (M, C) of $N_1 \|_A N_2$ is a *combination* of *ID*'s (M_1, C_1) of N_1 and (M_2, C_2) of N_2, if M is the disjoint union of M_1 and M_2 and the C_i can be obtained from C by projecting its elements to the first, second resp., component (and deleting the dummy $*$). Then one shows how the behaviours correspond: one can fire a transition (t_1, t_2) from (M, C) if and only if one can fire t_1 from (M_1, C_1) (provided it is a transition) and t_2 from (M_2, C_2) (provided it is a transition); the resulting *ID* of $N_1 \|_A N_2$ is again a combination of the resulting *ID*'s of N_1 and N_2. The same holds for transition starts, and in the case of refusal sets, the three nets fire sets that satisfy the inclusion in the fourth item of Definition 9.

This result can be extended to sequences by induction and then lifted to the action level. □

Now we are ready to show the characterization of the 'faster-implementation' relation, the main theorem of this paper.

Theorem 11. *Let N_1 and N_2 be testable nets. Then $N_1 \sqsupseteq N_2$ if and only if $ART(N_1) \subseteq ART(N_2)$.*

Proof. 'if': Let (O, d) be a timed test. Then $ART(N_1) \subseteq ART(N_2)$ implies $AL(N_1 \| O) \subseteq AL(N_2 \| O)$ by Theorem 10 and Proposition 8. Thus, if N_1 fails the test due to some $w \in AL(N_1 \| O)$, then so does N_2.

'only if': In this proof, superscripts are upper indices; e.g., v_1^2 is an item with two indices in the following and *not* the string $v_1 v_1$.

We assume $N_1 \sqsupseteq N_2$ and take some $v = v_1^1 \cdots v_{n_1}^1 w_1^{1+} \cdots w_{m_1}^{1+} X^1 \cdots v_1^l \cdots v_{n_l}^l w_1^{l+} \cdots w_{m_l}^{l+} X^l \in ART(N_1)$, where $l, n_i, m_i \in \mathbb{N}_0$. All asynchronous refusal traces of N_1 can be extended to end with a refusal set by Proposition 7 (iii), hence it is enough to consider traces of this form; in particular, $l \geqslant 1$. By Proposition 7 (i) and (ii), we may assume that the X_i only contain actions that occur as labels in N_1 or N_2, since all other actions can be refused in N_1 and N_2 any time; hence, all X_i are finite.

We construct a test (O, d) that a net fails if and only if it has v as asynchronous refusal trace. Then N_1 fails (O, d), hence N_2 does and we are done.

We choose $d = l + 1$ and define O as follows, where upper indices of the places and transitions correspond to the upper indices in v, i.e., to the round number. Before reading the general construction, have a look at Fig. 8 for the case that $v = abc^+d^+\{x\}ea^+\{c\}$; in this case $l = 2$.

Double circles in this figure indicate a place that 'owns' an additional ω-labelled transition: the name of the transition is the name of the place prefixed with ω, the preset of the transition only contains the respective place and the postset is empty. The transition itself is suppressed in the figure; observe that we could additionally suppress the transition ωc^{l+2} at the bottom of the figure and draw the place c^{l+2} as a double circle.

In the left 'column', the places p_i^k and transitions u_i^k correspond to the sequences $v_1^k \cdots v_{n_k}^k$ of complete actions in v, i.e., to ab and e. In the next 'column', the places c^k and transitions t_0^k form a kind

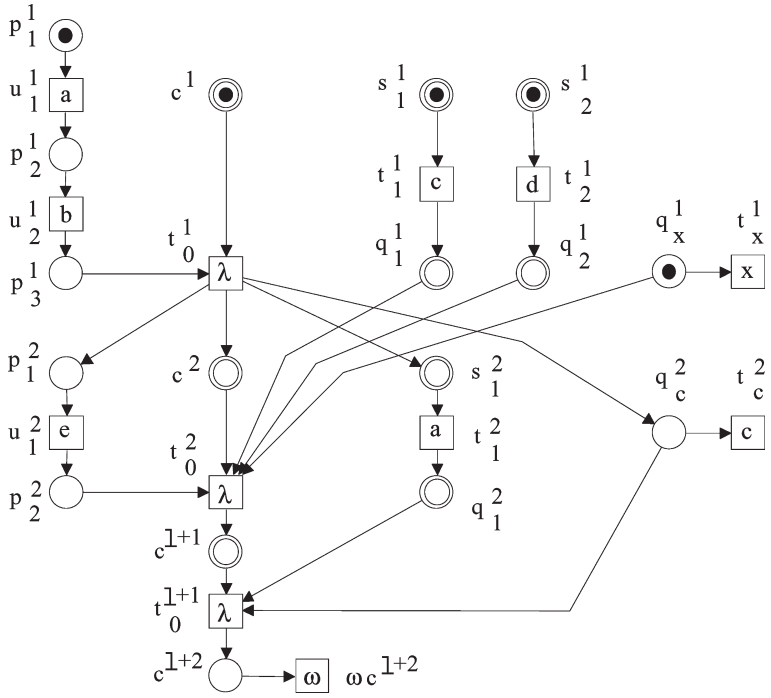

Fig. 8.

of clock, as we will see below. On the right hand side, the transitions t_i^k with $i \neq 0$ correspond to the sequences $w_1^{k+} \cdots w_{m_k}^{k+}$ of actions starting in the kth round, i.e., to c^+d^+ and a^+, and the transitions t_x^k correspond to the actions x that are refused in the kth round, i.e., $\{x\}$ and $\{c\}$.

$$
\begin{aligned}
S_O = &\{c^k \mid k = 1, \ldots, l+2\} \\
&\cup \{p_i^k \mid k = 1, \ldots, l;\ i = 1, \ldots, n_k + 1\} \\
&\cup \{s_i^k \mid k = 1, \ldots, l;\ i = 1, \ldots, m_k\} \\
&\cup \{q_i^k \mid k = 1, \ldots, l;\ i = 1, \ldots, m_k\} \\
&\cup \{q_x^k \mid k = 1, \ldots, l;\ x \in X^k\},
\end{aligned}
$$

$$
\begin{aligned}
T_O = &\{t_0^k \mid k = 1, \ldots, l+2\} \\
&\cup \{u_i^k \mid k = 1, \ldots, l;\ i = 1, \ldots, n_k\} \\
&\cup \{t_i^k \mid k = 1, \ldots, l;\ i = 1, \ldots, m_k\} \\
&\cup \{t_x^k \mid k = 1, \ldots, l;\ x \in X^k\} \\
&\cup \{\omega c^k \mid k = 1, \ldots, l+2\} \\
&\cup \{\omega s_i^k \mid k = 1, \ldots, l;\ i = 1, \ldots, m_k\} \\
&\cup \{\omega q_i^k \mid k = 1, \ldots, l;\ i = 1, \ldots, m_k\}.
\end{aligned}
$$

O has arcs for the following pairs, which are grouped into arcs for the 'clock', for the complete actions, for the starts and for the refusal sets:

- (c^k, t_0^k) for $k = 1, \ldots, l+1$,
- $(c^k, \omega c^k)$ for $k = 1, \ldots, l+2$,
- (t_0^k, c^{k+1}) for $k = 1, \ldots, l+1$,
- (p_i^k, u_i^k) and (u_i^k, p_{i+1}^k) for $k = 1, \ldots, l;\ i = 1, \ldots, n_k$,
- $(p_{n_k+1}^k, t_0^k)$ for $k = 1, \ldots, l$,
- (t_0^k, p_1^{k+1}) for $k = 1, \ldots, l-1$,
- (s_i^k, t_i^k) for $k = 1, \ldots, l;\ i = 1, \ldots, m_k$,
- $(s_i^k, \omega s_i^k)$ for $k = 1, \ldots, l;\ i = 1, \ldots, m_k$,
- (t_i^k, q_i^k) for $k = 1, \ldots, l;\ i = 1, \ldots, m_k$,
- $(q_i^k, \omega q_i^k)$ for $k = 1, \ldots, l;\ i = 1, \ldots, m_k$,
- (q_i^k, t_0^{k+1}) for $k = 1, \ldots, l;\ i = 1, \ldots, m_k$,
- (t_0^{k-1}, s_i^k) for $k = 2, \ldots, l;\ i = 1, \ldots, m_k$,
- (q_x^k, t_x^k) for $k = 1, \ldots, l;\ x \in X^k$,
- (q_x^k, t_0^{k+1}) for $k = 1, \ldots, l;\ x \in X^k$,
- (t_0^{k-1}, q_x^k) for $k = 2, \ldots, l;\ x \in X^k$.

Initially, the places c^1, p_1^1, s_i^1, $i = 1, \ldots, m_1$, and q_x^1, $x \in X^1$ are marked. The labelling is as follows:

- $l_O(t_0^k) = \lambda$ for $k = 1, \ldots, l+2$,
- $l_O(\omega c^k) = \omega$ for $k = 1, \ldots, l+2$,
- $l_O(u_i^k) = v_i^k$ for $k = 1, \ldots, l;\ i = 1, \ldots, n_k$,
- $l_O(t_i^k) = w_i^k$ for $k = 1, \ldots, l;\ i = 1, \ldots, m_k$
- $l_O(\omega s_i^k) = l_O(\omega q_i^k) = \omega$ for $k = 1, \ldots, l;\ i = 1, \ldots, m_k$,
- $l_O(t_x^k) = x$ for $k = 1, \ldots, l;\ x \in X_k$.

Some testable net N fails the above test $(O, l+1)$ if and only if $N \| O$ can perform an asynchronous trace w without a start of an ω and with $l+1$ complete rounds. According to the proof of Proposition 8, we can interpret the σ's in w as Σ's and w as an asynchronous refusal trace. According to Theorem 10 and Definition 9, such a w must be a composition of some asynchronous refusal trace of N and some $u \in ART(O)$, where no ω starts in u and u contains $l+1$ sets all containing ω, since ω is not synchronized; note that each testable net can always refuse ω by Proposition 7 (i) and (ii).

It is not hard to see that $v_1^1 \cdots v_{n_1}^1 w_1^{1+} \cdots w_{m_1}^{1+} (\Sigma - X^1) \cdots v_1^l \cdots v_{n_l}^l w_1^{l+} \cdots w_{m_l}^{l+} (\Sigma - X^l)\Sigma$ is such a u: O can fire $v_1^1 \cdots v_{n_1}^1$ and then start t_0^1 and perform $w_1^{1+} \cdots w_{m_1}^{1+}$ such that now only the places q_x^1 are marked, i.e., $\Sigma - X^1$ can now be refused. Similarly, the other rounds are performed, and after $\Sigma - X^l$, O can start t_0^{l+1} and refuse the final Σ. Hence, if a net N (like N_1 by Proposition 7 (iii)) can perform $v\emptyset$, then the unique composition of the above u and $v\emptyset$ is a suitable w and N fails the test.

We will now argue that the above u is essentially the only choice; the only possible variation is to make some refusal sets smaller according to Proposition 7(i) or to permute some action starts

according to Proposition 7(iv). From this we can conclude that a net N that fails $(O, l+1)$ must have the asynchronous refusal trace $v-$ or a variation of v where the refusal sets are larger or some sequences of action starts are permuted, but also then $v \in ART(N)$ again by Proposition 7(i) and (iv).

So how can some $u \in ART(O)$ without a start of ω and with $l+1$ sets all containing ω (or some $w \in ARFS(O)$ underlying such a u) look like? The key to answer this question is the 'clock' consisting of the places c^k and transitions t_0^k in the second 'column'. If some c^k is marked at some stage, this enables an ω-transition; if we want to avoid this transition and at the same time want to refuse ω, we have to start t_0^k in the present round; as a consequence, c^{k+1} will be marked in this or the next round. This observation implies that t_0^1 must start in the first round and end at the latest in the second; thus, t_0^2 must start at the latest in the second round and so on. In general, t_0^k must start at the latest in the kth round and end at the latest in the $(k+1)$st round.

In particular, t_0^{l+1} must end at the latest in the $(l+2)$nd round; should it end earlier, this would enable ωc^{l+2} and we cannot refuse ω in the last set of u without performing an ω. Hence, t_0^{l+1} must end in the $(l+2)$nd round and start in the $(l+1)$st round. By the above argument about marking some c^k, this means that t_0^l must end in the $(l+1)$st round and start in the round before. In general, t_0^k must start in the kth round and end in the $(k+1)$st round.

The first consequence is that p_1^k gets marked in the kth round; to enable t_0^k, the sequence $v_1^k \cdots v_{n_k}^k$ must be fired in the kth round. The second consequence is that the s_i^k get marked in the kth round; thus, t_i^k cannot be started before the kth round, but to avoid the firing of ωs_i^k, t_i^k must at least start now. If t_i^k ended in the kth round, then q_i^k would be marked in this round, but would only be emptied in the next; thus, ω would occur or it would not be refused. Therefore, t_i^k must end in the $(k+1)$st round and $w_1^{k+} \ldots w_{m_k}^{k+}$ is performed in some order. The third consequence is that the q_x^k get marked in the kth round and will be emptied in the $(k+1)$st round by t_0^{k+1}. Thus, no t_x^k must be fired in u; the only places marked throughout the kth round are the places q_x^k and the maximal refusal set is $\Sigma - X^k$. At the end of the last round, t_0^{l+1} has started, no place is marked and Σ can be refused.

Thus, there is no real choice for u and if a net N like N_2 fails the test $(O, l+1)$, then $v \in ART(N)$, which concludes the proof. □

Note that, according to this theorem, a faster system has fewer refusal traces; we can see such a trace as a witness of slow behaviour.

Analogously, we can characterize the 'm-implementation' relation.

Theorem 12. *N_1 is an m-implementation of N_2 if and only if $MRT(N_1) \subseteq MRT(N_2)$.*

Proof. Similar to the last proof. The construction of O has to be modified simply by omitting the places p_i^k and the transitions u_i^k. □

With the characterization of our testing preorder in Theorem 11, we get immediately that it is a precongruence; the corresponding ART-equivalence is fully abstract, i.e., it makes just the necessary distinctions if we want to distinguish nets with different asynchronous behaviour and want to work with parallel composition.

Corollary 13.
(i) *The relation $\sqsupseteq$ is a precongruence with respect to parallel composition.*
(ii) *ART-equivalence is fully abstract with respect to asynchronous equivalence and parallel composition, i.e. it is the coarsest congruence for parallel composition that respects asynchronous equivalence.*

Proof.
(i) Follows from Theorems 10 and 11.
(ii) Theorem 10 and Proposition 8 show that *ART*-equivalence is a congruence that respects asynchronous equivalence. If $ART(N_1) \neq ART(N_2)$, then the proof of Theorem 11 exhibits a net O such that $AL(N_1 \| O) \neq AL(N_2 \| O)$. (If N_1 or N_2 contain the special action ω, then its rôle in O must be played by some other action a not occurring in N_1 or N_2; consider $AL(N_i \|_{\Sigma-\{a\}} O)$ in this case.) □

Analogously we have the following corollary, where Part (ii) is a slight generalization of a result in [27] to nets with internal transitions.

Corollary 14.
(i) *The m-implementation relation is a precongruence with respect to parallel composition.*
(ii) *MRT-equivalence is fully abstract with respect to maximal-step equivalence and parallel composition.*

We close this section with a remark on may-testing. Our timed testing is a form of must-testing; as our characterization of the faster-implementation relation as *ART*-inclusion shows, it is different from classical must-testing. We have already explained that the corresponding may-testing is the same as classical may-testing, i.e. the may-implementation preorder is reverse language inclusion. Combining may- and must-testing, we would arrive at a preorder which is the intersection of *ART*-inclusion and language equivalence. This is in complete analogy with the classical approach, just as the result that this combined preorder strictly refines *ART*-inclusion: The empty net is clearly not language equivalent to the net N in Fig. 9, but it is a faster implementation of N; the empty net can only repeatedly refuse arbitrary sets, and N can immediately start the internal transition and do the same.

5. Further properties of *ART*-semantics

It is not always easy to check the faster-implementation relation, i.e., inclusion of *ART*-semantics, even for simple examples. Hence, it is advisable to compare nets in a structured way, e.g., by using forward simulations; see [16] for a survey on this and similar techniques. Usually, in a forward simulation one system simulates the actions of another; here, it also simulates the action starts and refusal sets –

Fig. 9.

observe that the occurrence of a refusal set corresponds to the end of a round and changes the state of the system.

Definition 15. For nets N_1 and N_2, a relation R between some ID's of N_1 and some of N_2 is a *(forward) simulation* from N_1 to N_2 if the following hold:

(i) $(ID_{N_1}, ID_{N_2}) \in R$

(ii) If $(ID_1, ID_2) \in R$ and $ID_1[t\rangle_r ID_1'$ or $ID_1[X\rangle_r ID_1'$, then for some ID_2' with $(ID_1', ID_2') \in R$ we have $ID_2[l_1(t)\rangle\!\rangle_r ID_2'$ or $ID_2[X\rangle\!\rangle_r ID_2'$. Observe that these moves from ID_2 to ID_2' may involve several transitions. Similary, $ID_1[t^+\rangle_r ID_1'$ implies that for some ID_2' with $(ID_1', ID_2') \in R$ we have $ID_2[l_1(t)^+\rangle\rangle_r ID_2'$ for visible t and $ID_2[\rangle\!\rangle_r ID_2'$ for internal t.

Analogously, we define an *m-simulation* using $[t^+\rangle_{mr}$ and $[l_1(t)^+\rangle\!\rangle_{mr}$, etc. in the above condition.

It is not difficult to show the following theorem, which we will apply below and in Section 6; see [16] for a proof of a similar theorem.

Theorem 16. *If there exists a simulation (m-simulation) from N_1 to N_2, then N_1 is a faster implementation (m-implementation) of N_2.*

The faster-implementation relation is via the *ART*-characterization the same as the inclusion of some formal languages. A simulation is a very useful tool to show such an inclusion; but inclusion might hold even if no simulation exists, see [16].

Recall that a round consists of a sequence of transitions/actions followed by a step, which is 'maximal w.r.t. the following refusal set'. But instead of this monolithic step we write a sequence of action/transition starts; this way we get more reachable ID's (those with current transitions), but as a pay-off we only have to check single transitions or transition starts in the above definition.

Of course, one could also define a 'step'-variant of the simulations; e.g., for the m-case, such a simulation would relate ID's such that, whenever a step together with a refusal set is performed from ID_1, then the same must be possible from ID_2 reaching a related ID. Actually, in this variant a simulation could exist even if there is no m-simulation according to our definition. Fig. 10 shows two nets that are m-implementations of each other: Since enabled internal transitions must start unless disabled, both nets start in the first round a λ- or a d-transition combined with an a-, b-, c-, d- or λ-transition; afterwards, everything can be refused and no action can be started anymore. Hence, an m-refusal trace has essentially only one round, and an m-simulation in the 'step'-variant from N_1 to N_2 simply relates the initial markings and also all markings that have at most one token on the respective left-hand-side place. According to our definition, no m-simulation from N_1 to N_2 exists: N_1 can start the upper d and then still has the choice between b and c; N_2 cannot simulate this.

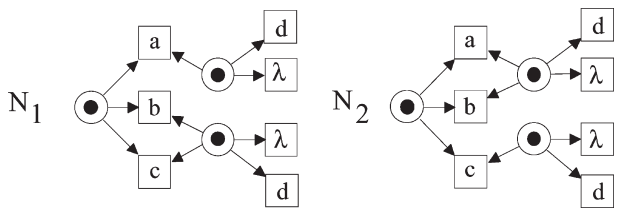

Fig. 10.

So far the developments of the asynchronous and the maximal-step case are very much in parallel. This will change when we look at the following three operations on nets:

Definition 17. Let N be a net. The *τ-prefix* $\tau.N$ of N is obtained by removing all tokens, adding a new marked place s and a new λ-labelled transition t with ${}^\bullet t = \{s\}$ and $t^\bullet = M_N$.

N' is an *elongation* of N, if it is obtained from N by choosing a transition t, adding a new unmarked place s and a new λ-labelled transition t' with ${}^\bullet t' = \{s\}$ and $t'^\bullet = t^\bullet$ and, finally, redefining $t^\bullet$ by $t^\bullet := \{s\}$.

Call a transition t of N *conflict-free*, if no reachable marking M with $M[t\rangle$ enables a transition t' with ${}^\bullet t \cap {}^\bullet t' \neq \emptyset$. Now N' is an *internal sequentialisation* of N, if it is obtained from N by choosing two internal, conflict-free transitions t and t' and adding a new marked place s to the pre- and postsets of t and t'.

One might expect that N and $\tau.N$, but also N and an elongation N' or an internal sequentialisation N'' are essentially equivalent – with N being faster: $\tau.N$ and N' have an additional internal transition, which might take time. In N'', t and t' can only fire in sequence, which would take more time than a possible parallel execution in N. Internal sequentialisation can be applied, e.g., if t and t' model the conflict-free transport of data; we will see a similar situation in Section 6. We have the following results, which recommend the asynchronous testing approach.

Proposition 18. *N is a faster implementation, but in general not an m-implementation, of $\tau.N$, of each elongation N' and of each internal sequentialisation N''.*

Proof. $\{(ID, ID) | ID \text{ is reachable in } N\} \cup \{(ID_N, ID_{\tau.N})\}$ is a simulation from N to $\tau.N$; the first move of N is matched by the new transition and the same move in $\tau.N$.

The identity relation is a simulation from N to N'; if t and t' are the transitions involved in constructing N', then t in N is matched by tt' in N', t^+ in N is matched by t^+ in N', and some X-move that 'closes' t is matched by Xt'.

Let N'' be obtained from N by adding s to the pre- and postsets of t and t'. We relate a reachable ID (M, C) of N to $(M \cup \{s\}, C)$ if $t, t' \notin C$, to (M, C) if either $t \in C$ or $t' \in C$, and to $(M \cup {}^\bullet t, C - \{t\})$ and $(M \cup {}^\bullet t', C - \{t'\})$ if $t \in C$ and $t' \in C$ where ${}^\bullet t$ and ${}^\bullet t'$ are formed in N. In the first two cases, each transition, transition start or refusal set of N is matched by the same item in N'' with the exception that t'^+ is ignored if $t \in C$ and vice versa. In the third case, a transition start in N is matched by the same item in N'' and a refusal set X in N is matched by Xt or Xt'; this is possible, since each transition t'' with $(M \cup {}^\bullet t)[t''\rangle$ in N'' is also enabled under the reachable marking $M + \sum_{u \in C - \{t\}} u^\bullet + {}^\bullet t$, which implies ${}^\bullet t'' \cap {}^\bullet t = \emptyset$ and $M[t''\rangle$ – the other case with $(M \cup {}^\bullet t')[t''\rangle$ is analogous.

Fig. 11 shows two nets a and $\tau.a$; we have $a^+ \in MRT(a) \backslash MRT(\tau.a)$. Similary, only the net on the left of Fig. 12 has the m-refusal trace $a^+ \emptyset b^+$.

Furthermore, only the net on the left of Fig. 13 has the m-refusal trace $\emptyset a$. □

Fig. 11.

Fig. 12.

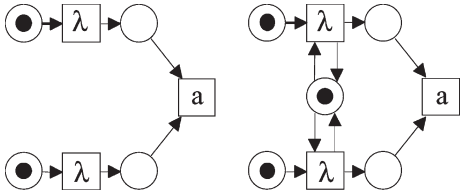

Fig. 13.

In most cases, N is strictly faster than the three types of nets listed in Proposition 18; at least, it is easy to find examples for this – see, e.g., Figs. 11 and 12. On the other hand, e.g., the net N in Fig. 14 is ART-equivalent to $\tau.N$.

It is interesting to note at this stage that the net N_1 in Fig. 15 is not a faster implementation of the net N_2 due to a^+b^+. This can be made plausible with the test net O in Fig. 16: $N_1 \| O$ can block the vital resources s and s' in the first round with a^+b^+ such that the time-critical activity of one λ- and the ω-transition, internal to O, takes place in the second and third round; hence, N_1 may fail $(O, 2)$ with $a^+b^+\sigma\sigma$.

In contrast, N_2 must satisfy $(O, 2)$; if in $N_2 \| O$ neither a nor a^+ occurs in the first round, the upper λ-transition at least starts in the first round and ω at the latest in the second – and the same holds for a occurring in the first round; this consideration applies analogously to b, if a^+ occurs in the first round. (The nets N_1 and N_2 are in fact incomparable under $\sqsupseteq$ due to the asynchronous refusal traces a^+b^+ and $a^+\{b\}b^+$.)

Still, one might want to regard N_1 as being faster than N_2; in order to find a justification for this view, one could consider to modify our approach such that $N_1 \sqsupseteq N_2$ does hold, e.g., by restricting the test nets under consideration; see also [14].

The nets of Fig. 17 demonstrate that the hypothesis of conflict-freeness in the definition of an internal sequentialisation is needed for Proposition 18: only the net on the left has the asynchronous refusal trace $\{a, b, c, d\}$.

We conclude this section by shortly considering some other operations. Fig. 18 shows $a + b$, the choice between a and b, on the left and $(\tau \|_\emptyset a) + b$ on the right; while a and $\tau \|_\emptyset a$ are ART- and MRT-equivalent, only $a + b$ has $\emptyset b$ as asynchronous and as m-refusal trace. (See, e.g. [26] for a definition of $+$.) Thus, these equivalences are not congruences with respect to choice. This phenomenon depends on initially enabled internal transitions and is shared by many other equivalences like weak bisimilarity.

Fig. 14.

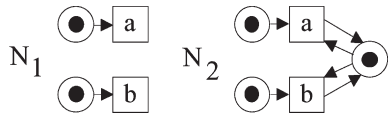

Fig. 15.

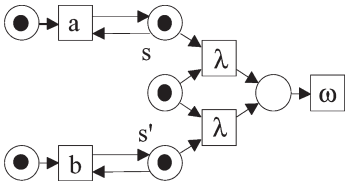

Fig. 16.

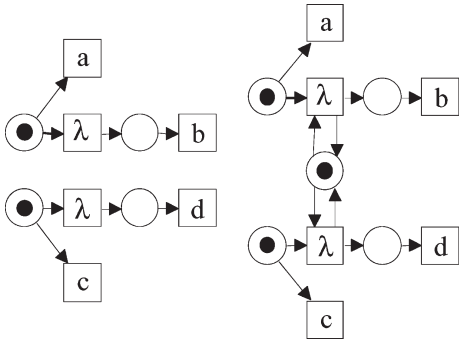

Fig. 17.

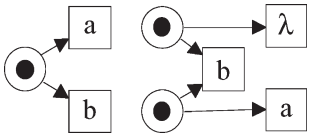

Fig. 18.

Another example that shows that $\sqsupseteq$ is not a precongruence is given in Fig. 6: although $a \sqsupseteq \tau.a$ (see Fig. 11), only $a + b$ (N_1 in Fig. 6) and not $\tau.a + b$ (N_2 in Fig. 6) has $\emptyset b$ as asynchronous refusal trace. This also shows that for an elongation that adds an internal transition in front of an existing transition we do not have a result like Proposition 18.

Definition 19. A *relabelling function* is a function $f : \Sigma \cup \{\lambda\} \to \Sigma \cup \{\lambda\}$ with $f(\lambda) = \lambda$ and $f(\Sigma) = \Sigma$. The *relabelling* $N[f]$ of N with relabelling function f is obtained from N by changing the labelling from l to $f \circ l$. *Hiding* $a \in \Sigma$ in N means changing all labels a to λ; it results in $N \backslash a$. *Restricting* $a \in \Sigma$ in N means deleting all a-labelled transitions; it results in N/a.

For these operations, we have the following precongruence- and, hence, congruence-results.

Theorem 20. $\sqsupseteq$ *and the m-implementation relation are precongruences with respect to hiding, relabelling and restriction.*

Proof. Hiding is the most interesting case, because similar results fail for ordinary failure or refusal trace semantics that do not consider divergence[1]; see [4] for a solution of this problem. $ART(N\backslash a)$ and $MRT(N\backslash a)$ can be constructed from those refusal traces in $ART(N)$ or $MRT(N)$ where a is contained in all refusal sets: delete all a and a^+ in these traces and replace the refusal sets by arbitrary subsets (possibly not containing a).

For restriction of a, consider those refusal traces that do not contain a or a^+ and add a to some refusal sets (– 'some' including the cases 'none' and 'all').

For relabelling it is enough to consider those functions that change some a to some b and leave all other actions unchanged. We can construct $ART(N[f])$ and $MRT(N[f])$ by changing in the refusal traces all a to b and all a^+ to b^+, removing b from the refusal sets that do not also contain a, and adding a to some refusal sets. □

6. Three implementations of a bounded buffer

In this section we will compare some implementations of a bounded buffer with respect to $\sqsupseteq$; this example has also been discussed in [1]. The first implementation *PIPE* is the usual sequence of buffers of capacity 1; the other two, *BUFFC* and *BUFFD*, use a buffer controller and an array as a circular queue to store the items. These two implementations differ only in a small detail making the buffer controller centralized in the first case and distributed (between input and output) in the second. Both variants are mentioned in [1], but only *BUFFC* is studied and shown to be faster than *PIPE*; and indeed, *BUFFC* and *BUFFD* are equivalent with respect to the efficiency preorder of [1]. This is a consequence of the interleaving approach taken in [1], which ignores that actions can be performed in parallel – which should take less time than performance one after the other.

One would expect that in reality *BUFFD* – being more distributed – is faster than *BUFFC*; also, an item should in the worst case take a long time to move through *PIPE*, so one might expect both *BUFF*-variants to be faster than *PIPE*. In our approach, it turns out that indeed $BUFFD \sqsupseteq BUFFC$ and $BUFFD \sqsupseteq PIPE$, but that – surprisingly – *BUFFC* is not (i.e., not always) faster than *PIPE* (nor the other way round). We will exhibit an asynchronous refusal trace as a witness of slow behaviour of *BUFFC* that *PIPE* cannot show; it will turn out that this slow behaviour indeed is a realistic possibility. Thus, our theory can help to find out facts about reality one might otherwise overlook.

For the rest of the section, we fix some $n \geqslant 4$ as capacity of the buffers. For simplicity, we assume that the items to be stored are from the set $\{0, 1\}$. We formally define *PIPE*, *BUFFC* and *BUFFD* as safe S/T-nets, the type of nets we study in this paper throughout. But in the figures we draw them as some sort of high-level net and hope that the translation will be clear: places are annotated with the type of tokens they store, and V stands for $\{\bullet, 0, 1\}$; arcs without annotation refer to ordinary tokens ($\bullet$), while we always have $x \in \{0, 1\}$.

[1] Here, ordinary failure semantics means the semantics as considered, e.g., in LOTOS or [9]. Another standard version of failure semantics, which does not have a problem with hiding, is given by stable failures, where only those refusal sets are considered that arise in states without internal moves, see, e.g. [3].

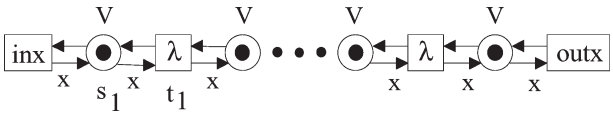

Fig. 19.

To explain a little the translation from the high-level net in Fig. 19 to the S/T-net defined below, consider, e.g., the first high-level place in Fig. 19: this is a 'cell' s_1 that stores a token to indicate that the cell is free or stores a value 0 or 1; it corresponds to the three places $(s_1, \bullet)$, $(s_1, 0)$ and $(s_1, 1)$ of the S/T-net. The first internal high-level transition t_1 in Fig. 19 moves a value 0 or 1 from the first to the second cell; it corresponds to the transition $(t_1, 0)$ with incoming arcs from $(s_1, 0)$ and $(s_2, \bullet)$ and to the transition $(t_1, 1)$ with incoming arcs from $(s_1, 1)$ and $(s_2, \bullet)$.

PIPE: The first implementation, *PIPE*, is shown in Fig. 19 and defined as follows:

$$S_{PIPE} = \{(s_i, v) \mid i = 1, \ldots, n,\ v \in V\}$$
$$T_{PIPE} = \{(t_i, x) \mid i = 0, \ldots, n,\ x \in \{0, 1\}\}$$

We have arcs for the following pairs with $i = 1, \ldots, n,\ x \in \{0, 1\}$:

$$((s_i, \bullet), (t_{i-1}, x)),\ ((t_{i-1}, x), (s_i, x)),\ ((s_i, x), (t_i, x)),\ ((t_i, x), (s_i, \bullet))$$

Initially, the places $(s_i, \bullet)$, $i = 1, \ldots, n$, are marked. The transitions (t_0, x) are labelled inx, $x \in \{0, 1\}$, the transitions (t_n, x) are labelled $outx$, $x \in \{0, 1\}$, and all other transitions are internal. □

The other two implementations, *BUFFC* and *BUFFD*, use one 'cell' for the recent input, one 'cell' for the next output and $n - 2$ 'cells' indexed from 0 to $n - 3$ for the other items in store. These 'cells' are used as a queue in a circular fashion; in *BUFFD*, *first* gives the index of the next item to be moved to the 'output cell', *last* gives the index of the next free 'cell' in the circular queue. Alternatively, *BUFFC* uses *first* and a 'variable' *length*, which gives the length of the circular queue. For the following, we put $I = \{0, \ldots, n - 3\}$ and let $\oplus$ and $\ominus$ denote addition and subtraction modulo $n - 2$.

BUFFC: For the translation of Fig. 20 to the following formal description of an S/T-net, observe, e.g., that the middle place represents the array: for each index $i \in I$, we have a cell with value $v \in V$, i.e.,

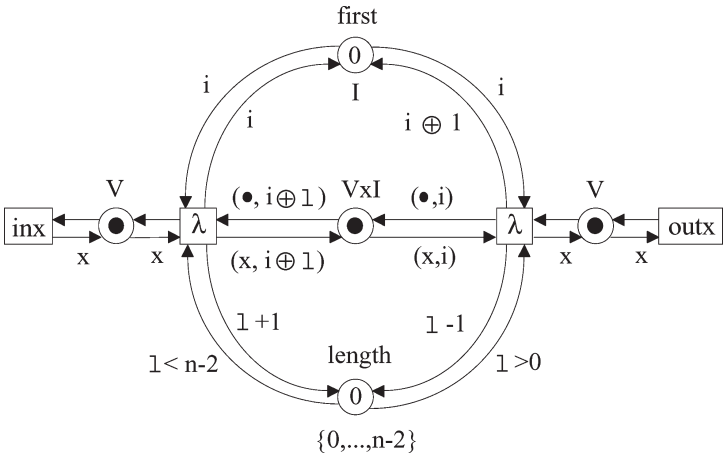

Fig. 20.

this place is translated to the places (s_i, v). Similarly, the variable *length* has value $l \in \{0, \ldots, n-2\}$ and corresponds to the places $length_l$. The internal transition on the left is the input controller; it reads the value i of *first*, the value l of *length* and moves the value x from the input cell s to the cell with index $i \oplus l$, provided this is free – which is indicated by a $\bullet$ in this cell; the transition correponds to the transitions $(t_{i,l}, x)$ in the S/T-net. On the right, we have the analogous situation for the output controller and the output cell s'. In general, each parameterized place, transition or arc in the formal definition corresponds one-to-one to a place, transition or arc in the figure.

$$\begin{aligned} S_{BUFFC} = &\{(s, v),\ (s', v) \mid v \in V\} \\ &\cup \{(s_i, v) \mid i \in I,\ v \in V\} \\ &\cup \{first_i \mid i \in I\} \\ &\cup \{length_l \mid l = 0, \ldots, n-2\}, \end{aligned}$$

$$\begin{aligned} T_{BUFFC} = &\{(t, x),\ (t', x) \mid x = 0, 1\} \\ &\cup \{(t_{i,l}, x) \mid i \in I,\ l = 0, \ldots, n-3,\ x = 0, 1\} \\ &\cup \{(t'_{i,l}, x) \mid i \in I,\ l = 1, \ldots, n-2,\ x = 0, 1\}. \end{aligned}$$

We have arcs for the following pairs with $x = 0, 1$ and $i \in I$:

$$\begin{array}{ll} ((s, \bullet), (t, x)),\ ((t, x), (s, x)) & \\ ((s, x), (t_{i,l}, x)),\ ((t_{i,l}, x), (s, \bullet)) & \text{with } l = 0, \ldots, n-3, \\ (first_i, (t_{i,l}, x)),\ ((t_{i,l}, x), first_i) & \text{with } l = 0, \ldots, n-3, \\ (length_l, (t_{i,l}, x)),\ ((t_{i,l}, x), length_{l+1}) & \text{with } l = 0, \ldots, n-3, \\ ((s_{i \oplus l}, \bullet), (t_{i,l}, x)),\ ((t_{i,l}, x), (s_{i \oplus l}, x)) & \text{with } l = 0, \ldots, n-3, \\ ((s', \bullet), (t'_{i,l}, x)),\ ((t'_{i,l}, x), (s', x)) & \text{with } l = 1, \ldots, n-2, \\ (first_i, (t'_{i,l}, x)),\ ((t'_{i,l}, x), first_{i \oplus 1}) & \text{with } l = 1, \ldots, n-2, \\ (length_l, (t'_{i,l}, x)),\ ((t'_{i,l}, x), length_{l-1}) & \text{with } l = 1, \ldots, n-2, \\ ((s_i, x), (t'_{i,l}, x)),\ ((t'_{i,l}, x), (s_i, \bullet)) & \text{with } l = 1, \ldots, n-2, \\ ((s', x), (t', x)),\ ((t', x), (s', \bullet)). & \end{array}$$

Initially, the places $(s, \bullet)$, $(s', \bullet)$, $first_0, length_0$ and $(s_i, \bullet),\ i \in I$, are marked. The transitions (t, x) are labelled $inx,\ x \in \{0, 1\}$, the transitions (t', x) are labelled $outx$, $x \in \{0, 1\}$, and all other transitions are internal. □

BUFFD: Fig. 21 can be translated similarly as above; here, the input controller accesses the input cell s, one cell of the array and only the variable *last*, and similarly for the output controller, the output cell s' and *first*.

$$\begin{aligned} S_{BUFFD} = &\{(s, v),\ (s', v) \mid v \in V\} \\ &\cup \{(s_i, v) \mid i \in I,\ v \in V\} \\ &\cup \{last_i, first_i \mid i \in I\} \end{aligned}$$

$$\begin{aligned} T_{BUFFD} = &\{(t, x),\ (t', x) \mid x = 0, 1\} \\ &\cup \{(t_i, x), (t'_i, x) \mid i \in I,\ x = 0, 1\} \end{aligned}$$

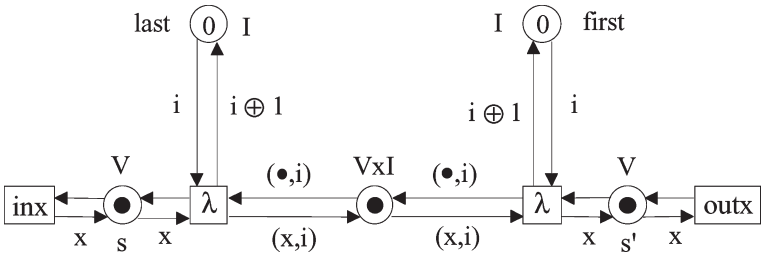

Fig. 21.

We have arcs for the following pairs with $x = 0, 1$ and $i \in I$:

$((s, \bullet), (t, x)),\ ((t, x), (s, x))$
$((s, x), (t_i, x)),\ ((t_i, x), (s, \bullet))$
$(last_i, (t_i, x)),\ ((t_i, x), last_{i\oplus 1})$
$((s_i, \bullet), (t_i, x)),\ ((t_i, x), (s_i, x))$
$((s', \bullet), (t'_i, x)),\ ((t'_i, x), (s', x))$
$(first_i, (t'_i, x)),\ ((t'_i, x), first_{i\oplus 1})$
$((s_i, x), (t'_i, x)),\ ((t'_i, x), (s_i, \bullet))$
$((s', x), (t', x)),\ ((t', x), (s', \bullet))$

Initially, the places $(s, \bullet)$, $(s', \bullet)$, $first_0$, $last_0$ and $(s_i, \bullet),\ i \in I$, are marked. The transitions (t, x) are labelled $inx,\ x \in \{0, 1\}$, the transitions (t', x) are labelled $outx,\ x \in \{0, 1\}$, and all other transitions are internal.

E.g. [1]) prefers *BUFFC* over *BUFFD*, presumably because the variable *length* helps to distinguish a full queue from an empty one. In *BUFFD*, $last = first$ if and only if the array is completely empty or completely full; observe that *BUFFD* only works because a token on $(s_i, \bullet)$ indicates that the cell s_i is free.

Comparing the three implementations, we first note that *PIPE* can be slow when transporting an item from input to output. Formally, since $n \geqslant 4$, we have $(in0)^+\emptyset\emptyset\emptyset\{out0\} \in ART(PIPE) \setminus (ART(BUFFC) \cup ART(BUFFD))$ as a witness of slow behaviour: item 0, input in the first round, can still not be delivered in the fourth round in *PIPE*. This only shows that sometimes – i.e., for some behaviour of the environment or user – *PIPE* is slower than *BUFFC* and *BUFFD*; it does not show that, e.g., *BUFFC* is always faster.

In *BUFFC*, all internal transitions access 'variable' *last* and *length*, hence input and output controller block each other. This has the surprising effect that *BUFFC* is in fact not faster than *PIPE*: we will exhibit a trace $w \in ART(BUFFC) \setminus ART(PIPE)$ as a witness of slow behaviour. It demonstrates that with some user behaviour the blocking leads to a slowdown.

This trace starts $(in0)\emptyset^{n-1}$; each refusal set $\emptyset$ requires the occurrence of at least one internal transition unless none is enabled; hence, after $(in0)\emptyset^{n-1}$ the first item 0 is in *BUFFC* stored in cell s', i.e., $(s', 0)$ is marked, and in *PIPE* in cell s_n. The next part is $(in1)\emptyset^{n-2}$, after which the second item 1 is in *BUFFC* in the queue in s_0 and in *PIPE* in s_{n-1}. The third part is $(out0)^+\emptyset$, i.e., 0 is removed from s', s_n resp. Now, in *PIPE* item 1 is moved to s_n in the next round and, henceforth, $out1$ is enabled – no

matter what else happens; in particular, in *PIPE* output cannot be blocked by input. But in *BUFFC*, $in0$ might be performed very fast, followed by a slow transport of 0 to s_1 at the end of the round; consequently, 1 is not moved from s_0 to s' in this round due to blocking, and $out1$ may be refused in the next round. Hence, for *BUFFC* – but not for *PIPE* – we can continue with $(in0)\emptyset\{out1\}$ giving $w = (in0)\emptyset^{n-1}(in1)\emptyset^{n-2}(out0)^+\emptyset(in0)\emptyset\{out1\} \in ART(BUFFC) \setminus (ART(PIPE)$. This trace shows: if the environment inputs two items and waits a long time (presumably doing something else), and afterwards requires output of the two items while inputting another one, then the second output can be blocked by the input in *BUFFC* but not in *PIPE*. This blocking in *BUFFC* and its absence in *PIPE* closely correspond to reality.

Actually, the above sort of behaviour can be repeated as long as there is space in the circular queue, and $out1$ is blocked for several rounds. Hence, we have $(in0)\emptyset^{n-1}(in1)\emptyset^{n-2}(out0)^+\emptyset((in0)\{out1\})^{n-3}$ $\{out1\}$ as another refusal trace as desired above, and it is in general even longer. In contrast, an m-refusal trace where $out1$ is blocked for several rounds is not possible; only in our asynchronous setting, the fast performance of input can block the output for an extended period.

In *BUFFD*, the buffer controller has an input and an output part, which communicate via the common store: the input part can store an item x in the circular queue only if the current cell s_{last} is marked as free with $\bullet$; the output part can remove an item from the circular queue only if the current cell s_{first} stores an item $x \in \{0, 1\}$. With this pattern of communication, the two parts can work in parallel and the input cannot block the output as above. We will show in the rest of this section that $BUFFD \sqsupseteq PIPE$ and $BUFFD \sqsupseteq BUFFC$, exhibiting two suitable simulations from *BUFFD* to *PIPE* and *BUFFC*. We will define these simulations more or less locally, mapping places and transitions of *BUFFD* to those of *PIPE* and *BUFFC*.

Remark. Note that the corresponding results do not hold if we use the maximal-firing rule: $(in0)^+\emptyset\emptyset\emptyset$ $(out0)^+ \in MRT(BUFFD) \setminus MRT(PIPE)$, and we also have $(in0)^+\emptyset\emptyset\emptyset(in1)^+\emptyset\emptyset(in0)^+(out0)^+\emptyset\emptyset$ $(in0)^+(out1)^+$ as an element of $MRT(BUFFD) \setminus MRT(BUFFC)$. The latter trace corresponds to the following behaviour: after the initial $(in0)^+\emptyset\emptyset\emptyset(in1)^+\emptyset\emptyset$ the value 0 is stored in s' and 1 in s_0; in the next round $(in0)^+(out0)^+\emptyset$ another 0 is moved to s while 0 is removed from s'; now only *BUFFD* can move 0 from s to s_1 and 1 to s' in the same round $\emptyset$ such that afterwards simultaneous in- and output is possible. But *BUFFD* should be faster than *BUFFC*, which is essentially an internal sequentialisation of *BUFFD*, see Proposition 18. This, I believe, is a strong argument in favour of the asynchronous approach of this paper compared to the maximal-firing approach.

We will describe some (in fact, all reachable) *ID*'s of *BUFFD* as triples $(\alpha_{i,x}, \beta'_{j,y}, w)$ with $x, y \in \{0, 1\}$, $\alpha, \beta \in \{a, b, c, d\}$; the meaning of such a triple is as follows:

- $\alpha = a$: $(s, \bullet)$ and $last_i$ are marked; note that $a_{i,0}$ and $a_{i,1}$ describe the same *ID*,
- $\alpha = b$: (t, x) is current and $last_i$ is marked,
- $\alpha = c$: (s, x) and $last_i$ are marked,
- $\alpha = d$: (t_i, x) is current.

Analogously:

- $\beta' = a'$: $(s', \bullet)$ and $first_j$ are marked; again $a'_{j,0}$ and $a'_{j,1}$ can be interchanged,
- $\beta' = b'$: (t', y) is current and $first_j$ is marked,
- $\beta' = c'$: (s', y) and $first_j$ are marked,
- $\beta' = d'$: (t'_j, y) is current.

Finally, $w \in \{0,1\}^*$ with length $|w| \in \{0,\ldots,n-2\}$ denotes the content of the queue. In principle, it fills the cells from j to $i \ominus 1$, while the cells from i to $j \ominus 1$ contain $\bullet$; but in case $\alpha = d$ cell i is being accessed and, hence, empty and in case $\beta' = d'$ cell j is empty. Thus, we require

- if $\beta' \neq d'$, then $|w| = i - j \pmod{n-2}$,
- if $\beta' = d'$, then $|w| + 1 = i - j \pmod{n-2}$,
- if $\alpha = d$ or $\beta' = d'$, then $|w| \neq n-2$,
- if $\alpha = d$ and $\beta' = d'$, then $i \neq j$.

To complete the *ID* of *BUFFD*, let $w = w_1 \ldots w_{|w|}$ with $w_k \in \{0,1\}$, $k = 1,\ldots,|w|$, and define m by $|w| + m + 2 = n-2$ if $\alpha = d$ and $\beta' = d'$, $|w| + m + 1 = n-2$ if either $\alpha = d$ or $\beta' = d'$, and $|w| + m = n-2$ otherwise. Then:

- if $\beta' \neq d'$, then $(s_{j\oplus k\ominus 1}, w_k)$, $k = 1,\ldots,|w|$, are marked,
- if $\beta' = d'$, then $(s_{j\oplus k}, w_k)$, $k = 1,\ldots,|w|$, are marked,
- if $\alpha \neq d$, then $(s_{i\oplus k\ominus 1}, \bullet)$, $k = 1,\ldots,m$, are marked,
- if $\alpha = d$, then $(s_{i\oplus k}, \bullet)$, $k = 1,\ldots,m$, are marked.

Note that ID_{BUFFD} can be described by $(a_{0,0}, a'_{0,0}, \lambda)$. The simulation relates the input part of *BUFFD* to the first 1-buffer of *PIPE*, the output part to the last 1-buffer and the circular queue to the last cells of *PIPE* up to s_{n-1}.

To relate ID_D described by $(\alpha_{i,x}, \beta'_{j,y}, w)$ to an ID_P of *PIPE* we define a partial function $f : S_{BUFFD} \cup T_{BUFFD} \to S_{PIPE} \cup T_{PIPE}$ as follows, see Figs. 22 and 23:

- $f(s,v) = (s_1, v)$, $f(s',v) = (s_n, v)$, $v \in V$,
- $f(s_{j\oplus k}, z) = (s_{n-1-k}, z)$ for $z \in \{0,1\}$ and $k = 0,\ldots,n-3$,

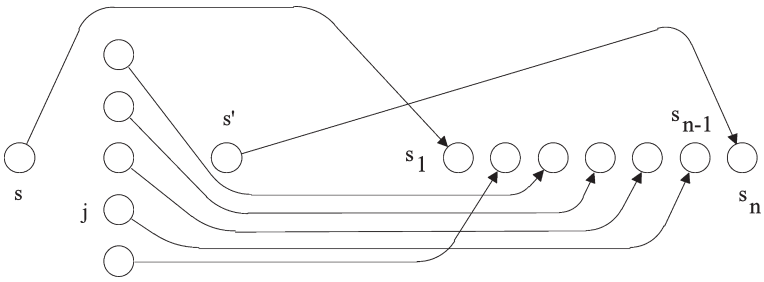

Fig. 22. Mapping of 0,1-contents.

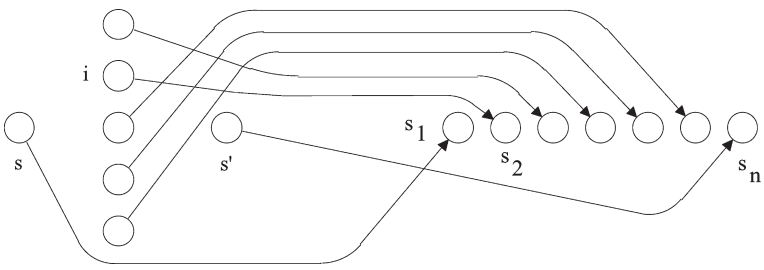

Fig. 23. Mapping of $\bullet$-contents.

- $f(s_{i\oplus k}, \bullet) = (s_{2+k}, \bullet)$ for $k = 0, \ldots, n-3$,
- $f(t, z) = (t_0, z)$, $f(t_i, z) = (t_1, z)$, $f(t'_j, z) = (t_{n-1}, z)$ and $f(t', z) = (t_n, z)$, $z \in \{0, 1\}$.

In ID_P a place $f(p)$ is marked if p is, a transition $f(t)$ is current if t is. This relates ID_{BUFFD} to ID_{PIPE}.

We check the transitions of *BUFFD*. If (t, x') or $(t, x')^+$ can fire, we have $\alpha = a$ and w.l.o.g. $x = x'$; hence, (t_0, x) or $(t_0, x)^+$ can fire, too. This firing gives $(c_{i,x}, \beta'_{j,y}, w)$ or $(b_{i,x}, \beta'_{j,y}, w)$ and the related *ID*'s of *PIPE*. If $M_D[(t_{i'}, x')\rangle$, then $\alpha = c$, $i' = i$ and $x' = x$; furthermore $(s_i, \bullet)$ is marked under ID_D, hence $(s_2, \bullet)$ is marked under ID_P and we conclude that $M_P[(t_1, x)\rangle$. Thus, if $(t_i, x)^+$ fires in *BUFFD*, then $(t_1, x)^+$ fires in *PIPE*, and we get to the *ID* described with $d_{i,x}$ and the related *ID* of *PIPE*. If (t_i, x) fires in *BUFFD*, then we have to move x to the end of w, i.e., we fire $(t_1, x)(t_2, x) \ldots (t_k, x)$ where $k = n - 2 + j - i$ if $i \geqslant j$ and $k = j - i$ if $i < j$. We end up with $(a_{i\oplus 1,x}, \beta'_{j,y}, wx)$ and the related *ID* of *PIPE*.

The treatment for (t', y') and $(t'_{j'}, y')$ is similar.

Finally, we have to consider the occurrence of a refusal set X; we can restrict attention to $X \subseteq \{in0, in1, out0, out1\}$. If X occurs, no internal transition may be enabled; note that (t_1, x), (t_{n-1}, x), $x \in \{0, 1\}$ are the only internal transitions of *PIPE* that could be enabled under M_P. If $\alpha \neq c$ and $\beta' \neq a'$, then no internal transitions are enabled under M_D and M_P; in both nets, X may contain $in0$ and $in1$ if $\alpha \neq a$ and $out0$ and $out1$ if $\beta' \neq c'$. The *ID* changes from $d_{i,x}$ to $a_{i\oplus 1,x}$ and x is appended to w or from $b_{i,x}$ to $c_{i,x}$ or stays at $a_{i,x}$; it also changes from $d'_{j,y}$ to $c'_{j\oplus 1,y}$ or from $b'_{j,y}$ to $a'_{j,y}$ or stays at $c'_{j,y}$. We get a corresponding change in *PIPE* if we fire X and

- move w by one position if $\beta' = d'$, i.e., we fire $(t_{n-2}, x_{n-2})(t_{n-3}, x_{n-3}) \ldots (t_k, x_k)$ for suitable $x_l \in \{0, 1\}$ where $k = n - 1 - |w|$,
- move x to the end of w if $\alpha = d$, i.e., we fire $(t_2, \bullet) \ldots (t_k, \bullet)$ where $k = n - 2 - |w|$.

Furthermore, some X can occur if $\alpha = c$ and all cells $s_0, \ldots, s_{n-3}$ are filled or being accessed by t'_j or if $\beta' = a'$ and $w = \lambda$; these cases are treated similarly.

We now turn to the simulation from *BUFFD* to *BUFFC*. To relate ID_D described by $(\alpha_{i,x}, \beta'_{j,y}, w)$ to an ID_C of *BUFFC* define $l = |w|$ if $\beta' \neq d'$ and $l = |w| + 1$ if $\beta' = d'$; define a partial function f that maps (s, v), (s', v), (s_i, v), $first_j$, (t, x') and (t', x') to themselves, and (t_i, x') to $(t_{i,l}, x')$ and (t'_j, x') to $(t'_{j,l}, x')$ for $v \in V$, $x' \in \{0, 1\}$. In ID_C a place $f(p)$ is marked if p is marked in ID_D and a transition $f(t)$ is current if t is current in ID_D with the following exceptions:

- $first_j$ is only marked if $\alpha \neq d$ and $\beta' \neq d'$; in this case, also $length_l$ is marked,
- if $\alpha = d$ and $\beta' = d'$, then either $(t'_{j,l}, y)$ is not current and $(s', \bullet)$ and (s_j, y) are marked or (t_i, x) is not current and $(s, \bullet)$ and (s_i, x) are marked.

This way, ID_{BUFFD} is related to ID_{BUFFC}. In general, the occurrences of transitions or transition starts are simulated by their images under f and a refusal set occurring under ID_D can occur under ID_C as well. There are the following exceptions:

- If $\alpha = d$, $\beta' = a'$ and $(t'_j, y)^+$ fires, then all (s, v), $length_k$ and (s', x'), $x' \in \{0, 1\}$ are unmarked in *BUFFC* and, thus, no transition of *BUFFC* is enabled. The resulting *ID* of *BUFFD* is $(d_{i,x}, d'_{j,y}, w')$ with $w'y = w$; it is also related to ID_C, so simulation is obtained by doing nothing. The next occurrence in *BUFFD* is an arbitrary refusal set, and this can also occur in *BUFFC*; after this, $(t'_{j,l}, y)$ fires to satisfy the simulation requirement.

- The case that $\alpha = c,\ \beta' = d'$ and $(t_i, x)^+$ fires is treated analogously.

Together, the above considerations show:

Theorem 21. *BUFFD is a faster implementation of PIPE and BUFFC, but no other $\sqsupseteq$-relations hold for the three variants of a bounded buffer. BUFFD is not an m-implementation of PIPE or BUFFC.*

7. Conclusion

We have developed a testing scenario for the worst-case efficiency of asynchronous systems. The resulting testing preorder can be characterized with some kind of refusal traces and satisfies some properties which make it attractive as a faster-than relation. We have applied the approach to compare three implementations of a bounded buffer – *BUFFC*, *BUFFD* and *PIPE*. In this approach, discrete time is used; it has been shown in [14] that using dense time one can arrive at the same faster-than relation. See, e.g. [5] for further developments of the approach presented in this paper.

In the approach of [1], *BUFFC* and *BUFFD* are equivalent, and *BUFFC* is shown to be faster than *PIPE* in [1]. This approach is based on some bisimulation-type preorder; visible actions are regarded as instantaneous and costs are measured as the number of internal actions; hence, [1] presents an interleaving approach, which disregards the parallel execution of actions. This is taken into account in the present paper, and consequently *BUFFD* is strictly faster than *BUFFC* and *PIPE* while, quite surprisingly, the latter two are incomparable. In particular, this is an example where the present approach makes more distinctions than the one of [1]. On the other hand, the latter approach applied to ordinary transition systems (which are sequential) without internal transitions gives ordinary bisimulation, while *ART*-equivalence can be shown to coincide with ordinary refusal trace equivalence. Hence, the approach of [1] distinguishes some systems which are equivalent here. While [1] considers worst and best case behaviour, a modification is presented in [2] which concentrates on the worst case behaviour.

Faster-than relations are also presented in [6,8,12,19,20]. In the first two papers, the behaviour of systems is influenced by timing considerations, i.e., the systems are not asynchronous. In [19], also a bisimulation-type preorder is defined and actions are regarded as instantaneous; a unit-time-delay operator with a special treatment is introduced, which makes the comparison to our approach very difficult. Such an operator is also used in [12], where a testing scenario is developed based on the maximal progress assumption, which is suitable for synchronous systems.

Corradini et al. [6] studies a bisimulation-type preorder with time-consuming actions, where time-stamps are attached to actions reflecting some *local* time; hence, actions do not necessarily occur (or are observed) in the order given by these local time-stamps – again this is a very different idea and no relation to our approach is obvious.

Finally, a testing approach with time-consuming actions is presented in [8], where the systems and tests under consideration have to be restricted to arrive at a faster-than relation; ordinary transition systems are used as models and, as a consequence, parallel execution is a priori excluded. Natarajan and Cleaveland [20] take the idea of [1] that efficiency is the number of actions, i.e., in effect the number of internal actions, as the basis of a testing approach and uses the time bounds introduced in [27].

One could say that our approach fills a gap in the latter paper, namely it presents must-testing with time bounds for asynchronous systems – assuming a time bound for each action. In [27], the duration of an action may vary depending on the circumstances, which are determined by the test environment O. This idea could be combined with our approach, but this is expected to lead to a much heavier notation – compare how events have to be introduced in [27] to tie starts and ends of actions together; such an extension is left for future efforts.

A translation of the idea of this paper to a process algebra has been presented in [7,13]. An important advantage of process algebras is that they can be provided with complete axiomatizations as it is done e.g., in [1,20]; these can be very useful for verification, and our current aim is to find such a complete axiomatization also for the new process algebra.

References

[1] S. Arun-Kumar, M. Hennessy, An efficiency preorder for processes, Acta Inform. 29 (1992) 737–760.
[2] S. Arun-Kumar, V. Natarajan, Conformance: a precongruence close to bisimilarity, In: J. Desel (Ed.), Structures in Concurrency Theory, Workshop in Computing, Springer, Berlin, 1995, pp. 55–68.
[3] J.A. Bergstra, J.W. Klop, E.R. Olderog, Failures without chaos: a new process semantics for fair abstraction, in: M. Wirsing (Ed.), Formal Description of Programming Concepts III, North-Holland, Amsterdam, 1987, pp. 77–103.
[4] E. Brinksma, A. Rensink, and W. Vogler. Fair testing, in: I. Lee, S. Smolka (Eds.), CONCUR 95, Lect. Notes Comp. Sci. vol. 962, Springer, Berlin, 1995, pp. 313–327.
[5] E. Bihler, W. Vogler. Efficiency of token-passing MUTEX-solutions – some experiments, in: J. Desel et al. (Eds.), Applications and Theory of Petri Nets 1998, Lect. Notes Comp. Sci. vol. 1420, Springer, Berlin, 1998, pp. 185–204.
[6] F. Corradini, R. Gorrieri, M. Roccetti, Performance preorder and competitive equivalence, Acta Inform. 34 (1997) 805–835.
[7] F. Corradini, W. Vogler, L. Jenner, Comparing the worst-case efficiency of asynchronous systems with PAFAS, Acta Inform. 38 (2002) 735–792.
[8] R. Cleaveland, A. Zwarico, A theory of testing for real-time, in: Proceedings of the Sixth Symposium on Logic in Computer Science, IEEE Computer Society Press, Silver Spring, MD, 1991, pp. 110–119.
[9] R. De Nicola, Extensional equivalences for transition systems, Acta Inform. 24 (1987) 211–237.
[10] R. De Nicola, M.C.B. Hennessy, Testing equivalence for processes, Theoret. Comput. Sci. 34 (1984) 83–133.
[11] J. Davies, S. Schneider, A brief history of Timed CSP, Theoret. Comput. Sci. 138 (1995) 243–271.
[12] M. Hennessy, T. Regan, A process algebra for timed systems, Inform. Comput. 117 (1995) 221–239.
[13] L. Jenner, W. Vogler, Comparing the efficiency of asynchronous systems, in: J.-P. Katoen (Ed.), AMAST Workshop on Real-Time and Probabilistic Systems, Lect. Notes Comp. Sci., vol. 1601, 1999, Springer, Berlin, pp. 172–191. Revised full version as [7].
[14] L. Jenner, W. Vogler, Fast asynchronous systems in dense time, Theoret. Comput. Sci. 254 (2001) 379–422.
[15] N. Lynch, M. Fischer, On describing the behaviour and implementation of distributed systems, Theoret. Comput. Sci. 13 (1981) 17–43.
[16] N. Lynch, F. Vaandrager, Forward and backward simulations I: Untimed systems, Inform. Comput. 121 (1995) 214–233.
[17] N. Lynch, Distributed Algorithms, Morgan Kaufmann Publishers, San Francisco, 1996.
[18] P. Merlin, D.J. Farber, Recoverability of communication protocols, IEEE Trans. Commun. COM-24 (1976) 1036–1043.
[19] F. Moller and C. Tofts. Relating processes with respect to speed, in: J. Baeten, J. Groote (Eds.), CONCUR '91, Lect. Notes Comp. Sci., vol. 527, Springer, Berlin, 1991, pp. 424–438.
[20] V. Natarajan, R. Cleaveland. An algebraic theory of process efficiency, in: Proceedings of the 11th Annual Symposium Logic in Computer Science (LICS '96), IEEE, 1996, pp. 63–72.
[21] J.L. Peterson, Petri Net Theory, Prentice-Hall, Englewood Cliffs, NJ, 1981.

[22] G. Peterson, M. Fischer, Economical solutions for the critical section problem in a distributed system, in: Proceedings of the Ninth ACM Symposium Theory of Computing, 1977, pp. 91–97.
[23] C. Ramchandi, Analysis of asynchronous concurrent systems by timed Petri nets. Technical Report TR 120, Project MAC, MIT, 1974.
[24] W. Reisig, Petri Nets, EATCS Monographs on Theoretical Computer Science, 4, Springer, Berlin, 1985.
[25] G.M. Reed, A.W. Roscoe, Metric spaces as models for real-time concurrency, Mathematical Foundations of Programming Language Semantics, Lect. Notes Comp. Sci., 298, Springer, Berlin, 1987, pp. 331–343.
[26] W. Vogler, Modular Construction and Partial Order Semantics of Petri Nets, Lect. Notes Comp. Sci., 625, Springer, Berlin, 1992.
[27] W. Vogler, Timed testing of concurrent systems, Inform. Comput. 121 (1995) 149–171.