UNIVERSITÄT AUGSBURG

Efficiency of Asynchronous Systems That Communicate Asynchronously

Walter Vogler

Report 1999-6 Dezember 1999

INSTITUT FÜR INFORMATIK
D-86135 AUGSBURG

Efficiency of Asynchronous Systems That Communicate Asynchronously

Walter Vogler [*]

Dezember 1999

Abstract

A parallel composition is introduced that combines nets (regarded as system components) by merging so-called interface places; the novel feature is a flexible typing of these places, which formulates assumptions a component makes about its environment. Based on a testing scenario, a faster-than relation is defined and shown to support modular construction, since it is a precongruence for parallel composition, hiding and renaming. The faster-than relation is characterized without reference to tests, and this characterization is used to compare the temporal efficiency of some examples.

keywords: synthesis and behaviour of nets; testing; verification; timed nets

1 Introduction

In the testing scenario of De Nicola and Hennessy [DNH84], systems are compared by the service they provide for a possible user or environment; thus, systems are regarded as components that communicate with other components, and this *communication* is *synchronous*. Service is understood as functional behaviour, i.e. which actions are performed, and since there is no consideration of time, this *behaviour* is *asynchronous*. This classical approach has been developed further in [Vog95, Vog97, BV98] in order to compare also the temporal efficiency of asynchronous systems – using Petri nets as system models; naturally, synchronous communication corresponds to combining nets by merging transitions.

From the dual nature of Petri nets, it is particularly natural to consider also the composition by merging places, which corresponds to *asynchronous communication*. This form of composition has found maybe less attention, but see e.g. [Che91, Vog92, BDE93, Val94, Gom96, Kin97]; compare e.g. [dBKPR91, dBZ99] for the mostly recent interest in asynchronous communication in the process algebra world. In this paper, we will develop a form of efficiency testing as in [BV98] etc., but based on asynchronous communication, i.e. merging of places.

On the one hand, to give maximal freedom in the modular construction of nets, it is desirable to allow the merge of arbitrary nets, see e.g. [Che91] or [Vog92, Section 4.3];

[*]This work was partially supported by the DFG-project ‘Halbordnungstesten’.

on the other hand, in many applications it is sensible to designate in a component N the interface places to be merged as input or output places. The latter usually means a restriction for N *and* the environment that we want to compose with N: if p is e.g. an input place of N, then often N is not allowed to have arcs to p and the environment is not allowed to have arcs from p. (E.g. [Vog92, Section 4.4] only requires the first, while e.g. [Kin97] requires both.)

In this paper, a (to my best knowledge) novel flexible typing of the interface is suggested: we will designate an interface place p of a net N as input or output place *or both*. These types correspond to assumptions about the environment, which is allowed to have arcs to p or from p or both; hence, the environment may produce some input for N or consume some output from N or both. Intuitively, N only guarantees proper behaviour if the environment or user satisfies the respective assumption. There is no assumption on N, but p is of course also an interface place of the environment, which will have its own assumptions. This way, we cover the liberal approach with maximal freedom as the case where each interface place under consideration is an in- and output place; and we cover the so often useful restricted approach essentially as the case where each interface place is either an input or an output place.

In the classical testing approach, a system is an implementation if it performs in all environments, i.e. for all users, functionally just as successful as the specification. Here, we also take into account the efficiency of implementation and specification: success (indicated by marking an output place ω) has to be reached within a given time. Thus, we are interested in worst case behaviour (so-called must-testing). By definition, components of asynchronous systems work with indeterminate speeds, (i.e. time cannot be used to coordinate components); most often, this is interpreted as 'each component may work arbitrarily slow'. Under this interpretation, the worst case is simply that nothing is done for a long time, hence every test is failed and we do not have a sensible theory of testing.

As a way out, [BV98] assumes that each action is performed within some given time bound (or is disabled within this time). Such an upper time bound is a reasonable basis for judging the efficiency, also see e.g. [PF77, Lyn96]; since actions can also be performed arbitrarily fast, the components work with indeterminate relative speeds also under this assumption, and we have a valid theory for asynchronous systems. While [BV98] defines a firing rule where each transition has to fire within time 1, we will also have transitions that have to fire immediately, i.e. within time 0; this demonstrates generality of our results (keeping things simple at the same time) and enhances expressivity as discussed in Sections 2 and 6

Nets and parallel composition $|||$ with place typing as described above are introduced in Section 2. In Section 3, we give our timed firing rule using discrete time and define efficiency testing; we call a net N_1 faster than a net N_2, if it satisfies all tests that N_2 satisfies, i.e. if it exhibits at least the same functionality with at least the same efficiency as N_2 specifies. We show that $|||$ is 'almost associative', which is used for our first main result that faster-than is a precongruence for $|||$ and thus allows compositional reasoning. The analogous result for renaming and hiding is given in Section 4, where – as second main result – we characterize faster-than without referring to tests. We point out how this result can be adapted to settings where there are no 0-transitions or where use of interface places is restricted as discussed above;

then we use the characterization in Section 5 to compare some example nets. In the conclusions, we also discuss work in progress regarding the construction of safe nets using $|||$ or variations thereof.

I thank Elmar Bihler and Laurentiu Tiplea for our discussions about this paper.

2 Basic Notions of Petri Nets with Time Bounds and Interface

In this section, we introduce Petri nets (S/T-nets) which are extended with a function that assigns to each transition a time bound 0 or 1, and with an interface of input and output places as explained in the introduction; we define the usual basic firing rule and the parallel composition for such nets.

Thus, a *Petri net with time bounds and interface* $N = (S, T, W, \beta, I, O, M_N)$ (or just a *net* for short) consists of finite disjoint sets S of *places* and T of *transitions*, the function $W : S \times T \cup T \times S \to \{0, 1\}$ describing the *arcs*, the *time bound* $\beta : T \to \{0, 1\}$, the *input places* $I \subseteq S$, the *output places* $O \subseteq S$, and the *initial marking* $M_N : S \to I\!N_0$. We associate with N its set of *interface places* $P = I \cup O$; note that an interface place p might be in I and O at the same time. We assume that M_N is 0 on P. When we introduce a net N or N_1 etc., then we assume that implicitly this introduces its components, e.g. S, I and P or S_1, I_1 and P_1 etc.

For each $x \in S \cup T$, the *preset* of x is ${}^\bullet x = \{y \mid W(y, x) = 1\}$, the *postset* of x is $x^\bullet = \{y \mid W(x, y) = 1\}$. These notions are extended to sets as usual, e.g. ${}^\bullet X$ is the union of all ${}^\bullet x$ with $x \in X$. If $x \in {}^\bullet y \cap y^\bullet$, then x and y form a *loop*. A *marking* is a function $S \to I\!N_0$. We sometimes regard sets as characteristic functions, which map the elements of the sets to 1 and are 0 everywhere else; hence, we can e.g. add a marking and a postset of a transition or compare them componentwise.

Assumption: In all nets considered in this paper, places are not isolated (i.e. ${}^\bullet s \neq \emptyset$ or $s^\bullet \neq \emptyset$ for each $s \in S$). Roughly speaking, isolated places are not very useful – as discussed below – and would lead to some technical complications.

As usual, we draw transitions as boxes, places as circles and arcs (i.e. pairs (x, y) with $W(x, y) = 1$) as arrows. If a place s is on a loop with a transition t with $\beta(t) = 1$ and has no arc to or from any other transition, we will not draw s but place a dot into the box of t instead; if the box representing some $t \in T$ does not have a dot, but $\beta(t) = 1$, then we give the box a double line on its right side; in both cases, t is called a *1-transition.* If there is no dot and no double line, then $\beta(t) = 0$, and t is a *0-transition.* An interface place is indicated by writing its type(s) I, O or IO next to it; we speak of an *IO-place* in the latter case. For an example, see Figure 1, which is discussed below.

We now define the usual basic firing rule, which ignores timing and interface.

- A transition t is *enabled* under a marking M, denoted by $M[t\rangle$, if ${}^\bullet t \leq M$.

  If $M[t\rangle$ and $M' = M + t^\bullet - {}^\bullet t$, then we denote this by $M[t\rangle M'$ and say that t can *occur* or *fire* under M yielding the marking M'.

- This definition of enabling and occurrence can be extended to sequences as usual: a sequence w of transitions 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 $w = w't$, $M[w'\rangle M''$ and $M''[t\rangle M'$ for some marking M'' and transition t. If w is enabled under the initial marking, then it is called a *firing sequence*; the set of those is denoted by $FS(N)$.

- A marking M is called *reachable* if $M_N[w\rangle M$ for some $w \in T^*$. The net is *safe* if $M(s) \leq 1$ for all places s and reachable markings M.

Before we introduce the parallel composition $|||$, we define nets to be *isomorphic* if one can be obtained from the other by bijectively renaming the transitions and the places in $S \setminus P$, while preserving the structure (arcs, time bounds and initial marking). We regard isomorphic nets as equal. Hence, only the identity of the interface places is important – and whenever we have two nets N_1 and N_2, we can assume that they are disjoint outside their interfaces, i.e. $(S_1 \cup T_1) \cap (S_2 \cup T_2) = P_1 \cap P_2$.

Now, if we combine nets N_1 and N_2 with the *parallel composition* $|||$, then they run in parallel and communicate asynchronously over their common interface. To construct the composed net, we will merge the common interface places of N_1 and N_2. But first of all, we assume that N_1 and N_2 satisfy the above disjointness condition – otherwise, a suitable isomorphic renaming is performed automatically. Additionally, the typing of the interface places as input or output places formulates an assumption about the respective other component: e.g. N_1 may only put a token onto an interface place p of N_2, if p is an input place of N_2. Thus, $N_1 \,|||\, N_2$ is defined if and only if, for $\{i, j\} = \{1, 2\}$, we have that $p \in (P_i \setminus I_i) \cap P_j$ and $t \in S_j$ implies $W_j(t, p) = 0$ and that $p \in (P_i \setminus O_i) \cap P_j$ and $t \in S_j$ implies $W_j(p, t) = 0$.

If these conditions are satisfied, $N = N_1 \,|||\, N_2$ is obtained by componentwise union except for the interface. (In particular, we can take the union of the two time bounds since they are defined on disjoint sets; the same is true for the initial markings, since whenever they are both defined on some p, then this is an interface place where both markings are 0.) Additionally, $I = (I_1 \setminus P_2) \cup (I_2 \setminus P_1) \cup (I_1 \cap I_2)$ and $O = (O_1 \setminus P_2) \cup (O_2 \setminus P_1) \cup (O_1 \cap O_2)$.

Let us explain the treatment of the interface. The interface of N is more or less (see next paragraph) the union of the interfaces of N_1 and N_2, and the typing of an interface place p is correspondingly determined by N_1 or/and N_2. If p is e.g. in I_1 but not in O_1, then N_1 assumes that its user will never take tokens from p; the same assumption must be made by N. Otherwise, a user U of N that takes tokens from p would indirectly use N_1 and violate its assumption.

If p is e.g. only an input for N_1 and only an output for N_2, then there is no arc to p in N_1 and no arc from p in N_2; this corresponds exactly to the restricted approach to composition by merging places that we discussed in the introduction. Note that such a p will not be in the interface of N, which is sensible, since no user U could satisfy both assumptions of N_1 and N_2; p allows point-to-point messages from N_2 to N_1 – a very useful application of our approach.

As an example for our nets and the application of $|||$, consider Figure 1. *Prod* is a typical user; it will quickly (i.e. with a 0-transition) put two 'orders' onto its output

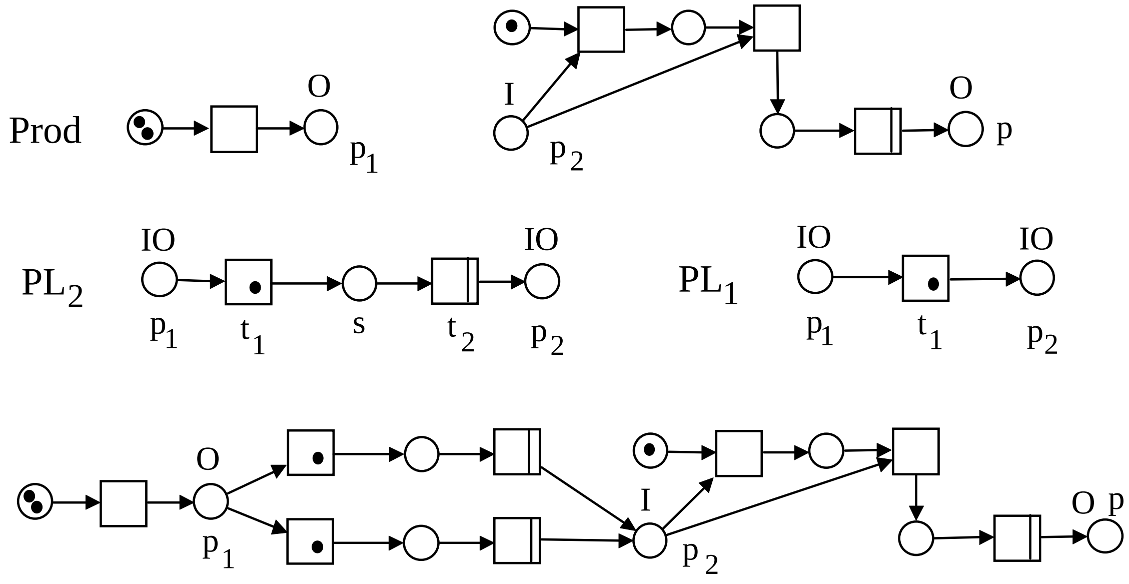

Figure 1

place p_1, and then it is willing to detect two 'products' on its input place p_2; after this, it would fire its 1-transition, i.e. 'use the products' for at most time 1, and then put a token onto p in order to signal satisfaction. Similarly, the special place ω is used in tests to signal success, i.e. satisfaction of the user.

The production line *PL2* can execute an 'order' placed on p_1 and put a 'product' onto p_2. Production has two stages (1-transitions) that each take up to time 1. The dot indicates a loop place, with the effect that in the first stage one order has to be processed after the other; one can run several orders through the second stage in parallel, taking up time 1 altogether; compare Definition 3.1. *PL1* is an improved version of *PL2*, needing only one stage; we certainly expect it to be faster.

The example also demonstrates the usefulness of our flexible interface typing. It is realistic to serve *Prod* with two production lines, i.e. to form the net *Prod* $|||$ (*PL2* $|||$ *PL2*) also shown in Figure 1; recall that the two copies of *PL2* are automatically made disjoint except for their interfaces. Intuitively, one would expect *PL2* $|||$ *PL2* to be faster than *PL2*. Note that both copies may take tokens from p_1; one copy of *PL2* would forbid the other one to do this, if p_1 would only be an input place of *PL2*; technically, *Prod* $|||$ (*PL2* $|||$ *PL2*) would not be defined. Thus, it is very reasonable to have IO-places as p_1 or similarly p_2.

Let us look at the case where we only consider nets where interface places are *either* input *or* output places. Let p be in I_1, I_2 and O_3 for the respective nets. Composing N_1 and N_2 requires that they have only arcs going to p; then, N_1 cannot be composed with e.g. N_3, i.e. p will ever be an input place with no ingoing arcs, which does not make much sense. Composing N_1 and N_3, on the other hand, requires that N_1 has no arcs going to p and N_3 has no arcs going out of p; thus, we are essentially in the restricted setting discussed in the introduction.

We mention in passing two other operations that can be useful in the construction of nets: *I- or O-hiding* an interface place p in a net N results in $N/^I p$ or $N/^O p$, which is obtained from N by removing p from I or O; *hiding* p is defined by $N/p = N/^I p/^O p$. *Renaming* an interface place p in a net N to $p' \notin P$ results in $N[p \to p']$, which is obtained from N by replacing p by p', which inherits p's arcs, marking and typing; if $p' \in S \setminus P$, it first has to be replaced isomorphically.

If we regard the combined production line $\mathit{PL2} \mid\mid\mid \mathit{PL2}$ as complete for the processing of orders, we should write it as $(\mathit{PL2} \mid\mid\mid \mathit{PL2})/^O p_1/^I p_2$. A producer that also uses two copies of *PL2*, but decides for each order which copy to use, could e.g. use *PL2* and $\mathit{PL2}[p_1 \to p'_1]$; products would still arrive at the unique place p_2.

3 Timed Behaviour of Asynchronous Systems and Testing

We will describe the asynchronous behaviour of a parallel system, taking into account the times at which things happen. The components of an asynchronous system vary in speed – but we assume that they are guaranteed to perform each enabled action within some finite given time; this upper time bound allows the relative speeds of the components to vary arbitrarily, since we have no positive lower time bound. Thus, the behaviour we define is truly asynchronous. In contrast to [BV98] etc., we do not assume that the upper time bound is 1 for all transitions; to demonstrate generality of the approach but keeping things simple at the same time, we allow 0 or 1 as bound. This will also allow to model transitions with arbitrary durations as discussed below.

The upper time bounds of transitions really correspond to firing intervals $[0, 1]$ and $[0, 0]$ attached to the transitions; instead, one could more generally attach arbitrary time intervals to the arcs from places to transitions as suggested in [SDdSS94]. Efficiency testing with synchronous communication has been studied for safe nets with such arc intervals in [Bih98], and it is shown that the faster-than relation based on testing is the same, no matter whether we work with continuous or discrete time (see also [Pop91]). The basic lemma for this result almost carries over to our setting (except that we use general S/T-nets); thus, for the time being, we will only use discrete time, which is easier to handle.

We will now define what we regard as timed behaviour; it will be convenient to use a formalism different from the ones used in the papers cited above. If a transition is enabled, it can always fire since transitions – only having an upper time bound – may fire very quickly; firing itself is instantaneous. If a 0-transition is enabled, it has to fire immediately, i.e. no time will pass before it has fired or has been disabled. If a 1-transition gets enabled, it may wait for one unit of time, and we write such a *time step* σ; to record how long transitions have been enabled, we record – additionally to the marking – how many tokens have an age of at least 1; then, a 1-transition has to fire immediately, if each place in its preset is marked with an 'old' token. This intuition is formalized as follows, where we assume that initially all tokens are old; if this is not adequate in some case, one can add a transition t that can fire once and produce the intended initial marking of fresh tokens.

Definition 3.1 A *timed marking* TM of a net is a pair (M, M^{old}) consisting of two markings of N with $M \geq M^{old}$. Again, introducing e.g. a timed marking TM_1 implicitly introduces M_1 and ${M^{old}}_1$ etc. The initial TM is $\mathit{TM}_N = (M_N, M_N)$.

We write $(M, M^{old})[\varepsilon\rangle(M', M^{old'})$ if one of the following cases applies:

1. $\varepsilon = t \in T$, $M[t\rangle M'$, $M^{old'} = M^{old} \ominus {}^{\bullet}t$, where $n \ominus m = \max(n - m, 0)$
2. $\varepsilon = \sigma$, $\forall t \in T : M[t\rangle \Rightarrow (\beta(t) = 1 \wedge \neg M^{old}[t\rangle)$, $M' = M^{old'} = M$

Generalizing this timed firing rule to sequences as above – together with a notion of reachable timed marking –, we define the set $\mathit{TFS}(N) = \{w \mid \mathit{TM}_N[w\rangle\}$ of *timed firing sequences* of N. For a timed firing sequence w, $\zeta(w)$ is the *duration*, i.e. the number of σ's in w. The behaviour in between two σ's is called a *round.* □

Part 2 of this firing rule ensures that every 1-transition that is enabled for one unit of time and every enabled 0-transition fires before time goes on, but according to Part 1 a 1-transition may also act faster. In fact, by only applying 1, we get $\mathit{FS}(N) \subseteq \mathit{TFS}(N)$; additionally, the occurrence of σ's only changes M^{old} while the firing of transitions only depends on M as usual. Hence, deleting the time steps from all sequences in $\mathit{TFS}(N)$ we get exactly $\mathit{FS}(N)$. This shows that, despite the upper time bounds, we still deal with the full complexity of asynchronous systems; we have simply enriched the asynchronous behaviour by some timing information in an orthogonal way.

Observe the following subtle point, where our consideration of asynchronous systems simplifies things a little: if a transition is given a choice, it always takes old tokens. One could also define a timed firing rule, where it might take a fresh one, i.e.: if $M(s) > M^{old}(s)$ – indicating the presence of a fresh token on s – and $t \in s^{\bullet}$ fires under TM, then we could allow that this firing leaves $M^{old}(s)$ unchanged. Such a rule would give sequences where the M^{old}-components are larger; the only consequence would be that we could apply Part 2 of this rule less often. But since we are never forced to apply 2 anyway, such an alternative firing rule would not give additional timed firing sequences. Thus, we can work with our simpler rule.

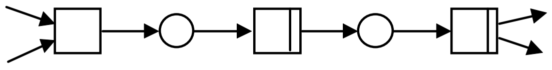

Figure 2

Quite generally, we can also model transitions with an arbitrary upper bound on their *duration.* E.g. Figure 2 shows a subnet where the first transition takes two tokens immediately once they are available, while the last transition produces two tokens after at most two time steps; thus, the subnet corresponds to a transition with maximal duration 2 with the described consumption and production. It is not clear whether we can model transitions that stay enabled up to time $n > 1$ and then fire instantaneously. See also Section 6 for expressivity.

We now define our testing scenario, where we compare nets by embedding them as components into testing environments and considering the behaviour of the resulting complete systems. Such a testing environment is something like a user that communicates with the embedded component; if after a while the user is satisfied with the service of the component, (s)he declares success by marking a special place ω – and this should happen within a prescribed period D of time even in the worst case. To be sure that we have seen everything that occurs up to time D, we only look at runs w with $\zeta(w) > D$.

Definition 3.2 A net is *testable* if none of its interface places is ω. A net is *a test net* if it has ω as output place with $\omega^\bullet = \emptyset$. A *timed test* is a pair (U, D), where U is a test net and $D \in \mathbb{N}_0$ (the *test duration*).

A testable net N *satisfies* a timed test (U, D) (N *must* (U, D)), if $N \,|||\, U$ is defined and each timed marking reached by some $w \in \mathit{TFS}(N \,|||\, U)$ with $\zeta(w) > D$ marks ω. For testable nets N_1 and N_2, we call N_1 a *faster implementation* of N_2 (or simply *faster than* N_2), $N_1 \sqsupseteq N_2$, if $N_1 \,|||\, U$ and $N_2 \,|||\, U$ are defined for the same test nets U and N_1 satisfies all timed tests that N_2 satisfies. $\sqsupseteq$ is also called the *efficiency preorder*. □

The efficiency preorder has two aspects: if $N_1 \sqsupseteq N_2$, then intuitively N_1 satisfies all U that N_2 specifies, i.e. it *functionally* implements N_2; and we speak of a *faster* implementation, since N_1 might also satisfy some specified test net within a shorter time. Note that N *must* (U, D) implies N *must* (U, D') for all $D' > D$; hence, if N_1 satisfies the same U as N_2 but with a different time D, this must be a shorter time.

Our timed firing rule allows some sort of Zeno-effect, i.e. we could have arbitrary long sequences with a fixed finite duration, in particular 0. This is of course not realistic, but in a way filtered out by the testing definition: a test fails only with a w that takes enough time, and since $\omega^\bullet = \emptyset$ in a test net, we can restrict attention to w that stop after the last σ. The filtering may have a misleading effect if some w with $\zeta(w) \leq D$ could not be extended to a ww' with $\zeta(ww') > D$; a system with such a time stop is ill-designed and should not be considered anyway. Note that time stops can only occur due to 0-transitions (or 1-transitions with empty preset, which should be avoided anyway since they act like 0-transitions).

From the above intuitive explanation, it should be clear that we can safely use N_1 instead of N_2, only if it is faster as just defined. We will validate our preorder by checking out examples below. But to be really useful, the faster-than relation should also support modular construction, i.e. it should be a precongruence: if $N_1 \sqsupseteq N_2$ and N_2 is a component of a system $N_2 \,|||\, N$, then replacing N_2 by N_1 should give a better system $N_1 \,|||\, N$, i.e. we should have $N_1 \,|||\, N \sqsupseteq N_2 \,|||\, N$.

Testing-based relations like the above faster-than relation usually give such a precongruence for the composition operator used in the definition of testing. [Vog92] discusses (actually for a slightly different setting, in particular considering synchronous communication and congruences instead of precongruences) that this is not necessarily so, but is always the case if the composition operator is commutative and associative (and the latter argument would also work here). Clearly, $|||$ is commutative, but it may violate associativity in somewhat pathological cases: Figure 3 shows three nets N_1, N_2 and N_3; forming $N_1 \,|||\, N_2$, the place p is removed from the interface, and then it is automatically replaced isomorphically when we construct $(N_1 \,|||\, N_2) \,|||\, N_3$ (top right net) as explained above. $N_1 \,|||(N_2 \,|||\, N_3)$ (bottom right net) is clearly different – also in behaviour.

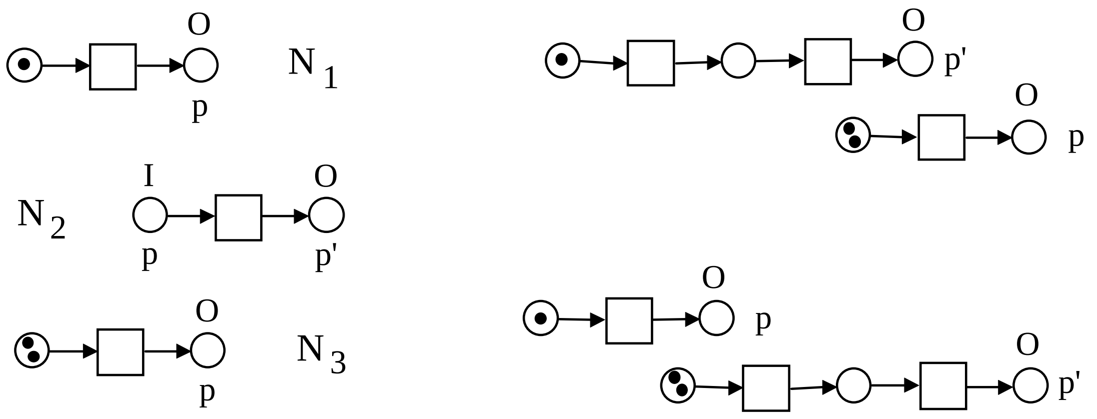

Figure 3

If we obtain N_3' from N_3 by adding p to I_3, then $(N_1 \,|||\, N_2) \,|||\, N_3'$ looks quite the same as $(N_1 \,|||\, N_2) \,|||\, N_3$; but $N_1 \,|||(N_2 \,|||\, N_3')$ is undefined, since there is an arc to $p \in I$ in $N = N_2 \,|||\, N_3'$, but $p \notin I_1$.

We will now give a sufficient criterion for associativity, and we will show how associativity can be failed. Together with a first result about nets that are comparable under our efficiency preorder, this will imply that $|||$ is 'associative enough' to prove that $\sqsupseteq$ is indeed a precongruence.

Theorem 3.3 *Let N_1, N_2 and N_3 be nets.*

i) If $\bigcap_{i=1}^3 P_i = (\bigcap_{i=1}^3 I_i) \cup (\bigcap_{i=1}^3 O_i)$, then $(N_1 \,|||\, N_2) \,|||\, N_3 = N_1 \,|||(N_2 \,|||\, N_3)$, which includes that one side is defined if and only if the other is.

ii) If $(N_1 \,|||\, N_2) \,|||\, N_3$ is defined and either $N_1 \,|||(N_2 \,|||\, N_3)$ is not defined or it is different from $(N_1 \,|||\, N_2) \,|||\, N_3$, then N_1, N_2 and N_3 have a common interface place that is only an output place of N_1 and only an input place of N_2 or vice versa; compare Figure 3.

iii) If N_1 is faster than N_2 (both testable), then N_1 and N_2 have the same input and the same output places, and for each common interface place p, we have that $p^\bullet \neq \emptyset$ in N_1 if and only if $p^\bullet \neq \emptyset$ in N_2, and similarly for the presets.

iv) Faster-than is a precongruence for testable nets.

Proof: i) If an interface place p belongs to at most two of the nets, it is easy to see that, with respect to p, definedness of the composition and classification as input and/or output place does not depend on the bracketing. If e.g. p does not belong to N_1, then composition with N_1 is certainly defined w.r.t. p and inherits arcs and classification from the other operand; hence, definedness, classification and arcs of p depend only on $N_2 \,|||\, N_3$.

Now let p be, say, an input place for all three nets; if it is also an output place for all three nets, then all intermediate and final results are defined and have p has in- and output place. If p is not an output place of e.g. N_1 (the other cases being similar), then definedness of the left-hand-side requires that there is no arc from p in N_2 and subsequently neither in N_3, whereas definedness of the right-hand-side requires that there is no arc from p in $N_2 \,|||\, N_3$, which also means that there is none in N_2 and none in N_3; thus, one side is defined w.r.t. p iff the other is, and the result will be the same w.r.t. p since the three nets merge on p; p will only be an input place of the final result.

ii) Assume the hypothesis holds; then, by i), there must be a common interface place p of all three nets that is not an input of all of them and not an output of all of them. If the claim fails, then p is without loss of generality an input place for N_1 and N_2, and only an output place for N_3. By the definedness of $(N_1 \,|||\, N_2) \,|||\, N_3$, the latter means that neither in N_1 nor in N_2 there are arcs to p. Furthermore, p is not an output place for one of N_1 or N_2, hence in the other there are no arcs from p. Thus, p is isolated in this other net, which is a contradiction to our general assumption.

iii) If p is only an input place of N_1, consider a test net U with interface consisting of ω and the IO-place p and with an arc from p; since $N_1 \,|||\, U$ is not defined, neither is $N_2 \,|||\, U$, and the only possible reason is that p is an interface place but not an output place of N_2. With a dual argument, we conclude that the interfaces of N_1 and N_2 (including the typing) coincide – with the possible exception that there might be some IO-place p of N_1 with $p \notin P_2$ (or vice versa).

So let p be such a place, and w.l.o.g. $p^\bullet \neq \emptyset$ by our assumption on nets. Consider a test net U with interface ω plus input place p. Again, composition with U is undefined for N_1, hence for N_2; we conclude that p must be an interface place of N_2, and thus the exception is in fact not possible.

The last consideration showed that also $p^\bullet \neq \emptyset$ in N_2. If we take U in this consideration without arcs from p, we could draw the same conclusion for $p \in P_1 \setminus O_1$. Analogous arguments conclude the proof.

iv) Let N_1 be faster than N_2, and let N be another testable net. From iii) we conclude that $N_1 \,|||\, N$ is defined iff $N_2 \,|||\, N$ is (which we assume now), and that composition with $N_1 \,|||\, N$ and $N_2 \,|||\, N$ is defined for the same test nets. Now let (U, D) be a test that $N_2 \,|||\, N$ satisfies; we have to show that $N_1 \,|||\, N$ satisfies the test as well.

Satisfaction depends on the behaviour of $(N_2 \,|||\, N) \,|||\, U$, $(N_1 \,|||\, N) \,|||\, U$ resp. If we can apply associativity in both cases, then N_2 satisfies $(N \,|||\, U, D)$ because satisfaction depends on the same net $N_2 \,|||(N \,|||\, U)$; thus; N_1 also satisfies $(N \,|||\, U, D)$, and we are done.

If associativity fails, we apply ii) to get a common interface place of N_1, N_2, N and U that w.l.o.g. is only an input place for N_1 and N_2 (apply iii)) and only an output place of N. Take a fresh place p' and $N_1' = N_1[p \to p']$, $N_2' = N_2[p \to p']$ and $N' = N[p \to p']$. Clearly, N_1' is faster than N_2' and composition with N' is defined for both these nets. Now $N_1 \,|||\, N$ and $N_1' \,|||\, N'$ are isomorphic, since p and p' are not in their interfaces. Thus, it suffices to consider N_1', N_2', N' and U; repeating this argument, we arrive at nets were associativity can be applied and are done. □

Note that by ii) $|||$ is almost associative, and the proof of iv) shows that with a suitable renaming we can always achieve associativity.

Remark: To prove this theorem, we have used our assumption that there are no isolated places. Possibly, one could show precongruence without this assumption; but here and later on, this would at least incur technical complications, which are quite unnecessary: isolated places could only make any difference in our setting, if they are interface places. Consider nets N_1 and N_2 with common interface place p that is isolated in N_1; let N_1' be obtained from N_1 by removing p. Comparing $N_1 \,|||\, N_2$ and $N_1' \,|||\, N_2$, p could have the effect that only the former is undefined – an effect which does not seem sensible –; otherwise, both compositions coincide except that p might be hidden from the in- or output in the former – an effect that one should achieve in a cleaner fashion with hiding.

4 Characterization of the Efficiency Preorder and a Proof Method

The efficiency preorder $\sqsupseteq$ formalizes observable difference in efficiency; referring to all possible tests, it is not easy to work with directly. Therefore, our aim is now to characterize $\sqsupseteq$ by only looking at the nets themselves that are compared. In classical testing [DNH84], such a characterization is based on failure pairs, which have just one so-called refusal set giving information on the final state of a run, while [BV98] uses refusal traces where repeatedly refusal sets give information on intermediate states of a run.

In a setting comparable to classical testing but using asynchronous communication, [Vog92, Section 4.3] uses a standard environment and then arrives at a refusal-type semantics that is much more involved than failure semantics. We will also use this standard environment, but pleasingly our characterization is a fairly simple refusal-type semantics similar to refusal traces (explained in more detail after Definition 4.1). This underlines the usefulness of the failure/refusal paradigm also for treating asynchronous communication, in contrast to [dBKPR91].

Definition 4.1 For a net N, N^{env} is obtained from N by adding the following *interface transitions*:

- for each $p \in I$, a fresh 1-transition p^+ and an arc from p^+ to p; and
- for each $p \in O$, a fresh 1-transition p^1, a fresh 0-transition p^0 and arcs from p to p^1 and p^0.

For timed markings (M, M^{old}) and $(M', M^{old'})$ of N^{env}, we define the *pr-firing* (pr stands for place refusal) $(M, M^{old})[\varepsilon\rangle_{pr}(M', M^{old'})$ if one of the following cases applies:

1. $\varepsilon = t \in T^{env}$, $M[t\rangle M'$, $M^{old'} = M^{old} \ominus {}^\bullet t$
(if $t \in \{p^0, p^1\}$ for some $p \in O$, we will actually write p^- instead of t in this case, since p^0 and p^1 are the same w.r.t. firing);
2. $\varepsilon = X \subseteq \{p^0, p^1 \,|\, p \in O\}$, $M' = M^{old'} = M$ and, for all $t \in T^{env}$, $M[t\rangle$ implies either $\beta(t) = 1 \wedge (\neg M^{old}[t\rangle \vee \exists p \in O : t = p^1 \notin X)$ or $\exists p \in O : t = p^0 \notin X$ or $\exists p \in I : t = p^+$; X is called a *refusal set.*

Generalizing this pr-firing rule to sequences as above, we define the set $PRS(N) = \{w \mid TM_N^{env}[w\rangle_{pr}\}$ of *pr-sequences* of N – where TM_N^{env} is the initial timed marking of N^{env}. For timed markings TM and TM' of N^{env}, we write $TM[v\rangle\rangle_{pr} TM'$, if v is obtained from some w by deleting all $t \in T$, such that $TM[w\rangle_{pr} TM'$ and w ends with a set; w is called the pr-sequence *underlying* the *pr-trace* v, and $PR(N)$ is the set of these pr-traces. The behaviour in between two refusal sets is called a *round.* □

In N^{env}, N is wrapped into a standard environment that might put/take tokens to/from the interface of N, depending on the typing. The pr-firing rule might look very complicated at first sight, but in fact it is not: it is very similar to the timed firing rule above, except that time steps are indicated by refusal sets instead of σ's. Occurrence of such a set is a 'partial σ'; it requires from the transitions of N the same as σ, but ignores the additional transitions p^1, p^0 not listed in the set and the additional p^+. That pr-traces always end with a set (in contrast to [BV98]) is related to the fact that (w.l.o.g.) the decisive behaviour for failing a test ends with a σ (see above), and it is required due to the possibility of time stops discussed in the previous section.

Observe that we only see interface transitions in the pr-traces, and that their existence depends on the interface typing; compare the second example in Section 5. We will show that our faster-than relation coincides with PR-inclusion.

Usually, the next step in an approach as ours would be to show that the PR-semantics of a composition can be constructed from the PR-semantics of the components; this would be used in the characterization proof and imply precongruence. Unfortunately, such a denotational construction has not been found as yet. But luckily, we already have a precongruence result, and the following proposition will be good enough to verify the characterization afterwards; it shows that we can find out enough about a composition from knowing one component and the PR-semantics of the other. We need two lemmas first.

Lemma 4.2 *Let N be a net, $t \in T$, $p \in I$ and $q \in O$. Then in N^{env}, $TM[tp^+\rangle_{pr} TM'$ implies $TM[p^+t\rangle_{pr} TM'$ and $TM[p^-t\rangle_{pr} TM'$ implies $TM[tp^-\rangle_{pr} TM'$.*

Proof: Obvious since p^+ only produces a token and p^- only takes a token; note that t and p^- will take old tokens if possible. □

Lemma 4.3 *Let N and U be nets such that $N_0 = N \,|||\, U$ is defined; let U_P be obtained from U by deleting P. Let TM_0 and TM'_0 be timed markings of N_0, let TM_P and TM'_P be their restrictions to the places of U_P, and TM and TM' their restrictions to the places of N^{env}. Let $t \in T_0$, such that ${}^\bullet t \cap P = \{p_1, \ldots, p_n\}$ and $t^\bullet \cap P = \{q_1, \ldots, q_m\}$ in case that $t \in T_U$.*

1. *If $t \in T$, then $TM_0[t\rangle TM'_0$ if and only if $TM[t\rangle_{pr} TM'$ and $TM_P = TM'_P$.*
2. *If $t \in T_U$, then $TM_0[t\rangle TM'_0$ if and only if $TM[p_1^- \ldots p_n^- q_1^+ \ldots q_m^+\rangle_{pr} TM'$ and $TM_P[t\rangle_{pr} TM'_P$.*

3. *$TM_0[\sigma\rangle TM'_0$ if and only if $M'_P = {M^{old}}'_P = M_P$ and there exists some X such that $TM[X\rangle_{pr} TM'$ and, for all $t \in T_U$, $M_P[t\rangle$ implies either $\beta_U(t) = 1 \wedge (\neg {M^{old}}_P[t\rangle \vee \exists p \in {}^\bullet t \cap P : p^1 \in X)$ or $\beta_U(t) = 0 \wedge \exists p \in {}^\bullet t \cap P : p^0 \in X$.*

Proof: By case analysis, with the following line of argument. (Also compare the similar [Vog92, 4.3.5], where the time step poses an additional complication here.)

We have grouped the common interface of N and U with N. Consequently, firing a transition of N depends on N only, while the state of U_P remains unchanged; firing a transition of U on the other hand depends on U_P but also on N as far as changes to the interface are concerned. The *effect* of a time step is easy to compare, but the somewhat tricky part are the conditions for *enabledness*: occurrence of X ensures that all transitions of N allow this time step; on the side of U_P, each transition allows a time step in U_P or the time step is justified by the interface places, and sufficient justifications are listed in X. More in detail, $p^1 \in X$ means that there is no old token on $p \in {}^\bullet t$, hence the 1-transition t allows a time step; and $p^0 \in X$ means that there is no token on $p \in {}^\bullet t$, hence the 0-transition t allows a time step. Thus, two partial time steps in N and U_P add up to a time step in N_0. $\square$

Proposition 4.4 *Let N_1, N_2 and U be nets such that $I_1 = I_2$, $O_1 = O_2$, $PR(N_1) \subseteq PR(N_2)$ and $N_1 \,|||\, U$ and $N_2 \,|||\, U$ are defined. Then, for each $w \in TFS(N_1 \,|||\, U)$ with $\zeta(w) = n$ reaching the timed marking TM_1 and ending with a σ, there exists a $v \in TFS(N_2 \,|||\, U)$ with $\zeta(v) = n$ reaching the timed marking TM_2 and ending with a σ such that TM_1 and TM_2 coincide on $S_U \setminus P_1$.*

Proof: Applying Lemma 4.3, w 'splits off' a $w_1 \in PRS(N_1)$ ending with a refusal set and some behaviour on the side of U; to the pr-trace $u \in PR(N_1) \subseteq PR(N_2)$ corresponding to w_1, we find an underlying $v_2 \in PRS(N_2)$ ending with a refusal set from which we can construct v stepwise with the above behaviour on the side of U, using again Lemma 4.3. Note that the timed marking on $S_U \setminus P_1$ changes the same way along w and v.

The only problem we could meet when constructing v is that application of 4.3.2 requires a suitable 'block' $p_1^- \dots p_n^- q_1^+ \dots q_m^+$ in v_2, but this sequence might actually be interspersed with transitions from T_2. In this case, we transform v_2 using Lemma 4.2 such that v_2 has the necessary 'blocks'. $\square$

Theorem 4.5 *Let N_1 and N_2 be testable nets. Then $N_1 \sqsupseteq N_2$ if and only if the following holds: $I_1 = I_2$, $O_1 = O_2$, and for each interface place p, we have that $p^\bullet \neq \emptyset$ in N_1 if and only if $p^\bullet \neq \emptyset$ in N_2, and similarly for the presets; furthermore, $PR(N_1) \subseteq PR(N_2)$.*

Proof: "if": Clearly, composition with N_1, N_2 resp., is defined for the same nets. Let (U, D) be a timed test. If N_1 fails the test, then due to a $w \in TFS(N_1 \,|||\, U)$ with $\zeta(w) > D$ ending with a σ such that ω is not marked afterwards. Proposition 4.4 gives some $v \in TFS(N_2 \,|||\, U)$ with which N_2 fails the test as well.

"only if": Theorem 3.3 iii) shows the first part. To prove the other part, we construct for each $w \in PR(N_1)$ a test that is failed by a testable net N if and only if $w \in PR(N)$; then N_1 fails this test, hence N_2 does, so $w \in PR(N_2)$ and we are done.

We sketch this construction by way of an example; the construction is akin to constructions in the case of synchronous communication except for the 0-transitions and the treatment of the last refusal set. For $w = p^+ q^- \{x^1, z^0\}\{z^1\} p^- \{x^0\}$, we choose $D = 3$ and the test net U shown in Figure 4. The interface of U consists of the omitted IO-places p, q, x and z, and the output place ω. A transition labelled e.g. p^+ (or ω) has an arc to p (or ω), a transition labelled e.g. p^-, p^0 or p^1 has an arc from p.

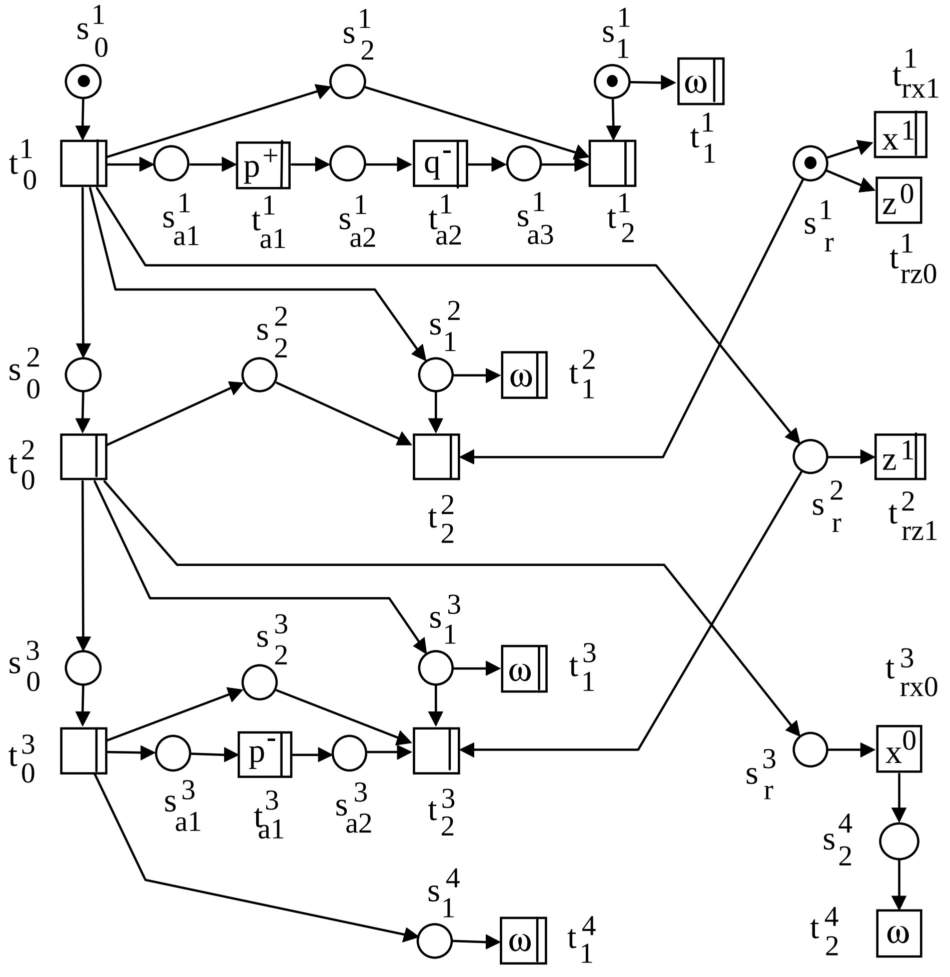

Figure 4

We will consider a possible timed firing sequence v that makes a net N fail the test $(U, 3)$ from the perspective of Lemma 4.3, i.e. we will consider what pr-trace N^{env} sees and what U has to do.

The subnet consisting of the places and transitions with lower index 0 act as a clock: since the token on s_0^1 is old initially, t_0^1 has to fire in the first round, t_0^2 at the latest in the second and t_0^3 at the latest in the third; actually, they must fire exactly in these rounds because otherwise, t_1^4 would fire at the latest in the third round and the test would not be failed. Hence, when v occurs, we have an old token on s_1^i and on s_r^i in the ith round, $i = 1, 2, 3$. Since we must not fire t_1^i in the ith round, we must fire t_2^i instead.

The first consequence is that, before t_1^2 fires, N sees exactly a token being put onto p and then a token being taken from q in the first round, i.e. p^+ and then q^- fire in N^{env}; analogously the other rounds can be treated.

The second consequence is that the $t_{r..}^i$, $i = 1, 2$, must not fire; hence, just before e.g. the first σ occurs, the token on s_r^1 is old and there cannot be a token on z or an old token on x. Thus, N^{env} sees the refusal set $\{x^1, z^0\}$ at the end of the first and $\{z^1\}$ at the end of the second round.

For the last round, the reason that t_{rx0}^3 must not fire, is different: if it fires in v, i.e. at the latest in the third round, then we cannot have a time step before the 0-transition t_2^4 fires, which would satisfy the test.

In consequence, N^{env} must perform the pr-trace w to fail the test $(U, 3)$, and the above analysis should also show how to fail the test, when N^{env} performs w. □

Observe that a faster system has less pr-traces, i.e. such a trace is a witness for slow behaviour, it is something 'bad' due to the refusal information it contains.

Also observe that the above proof is somewhat modular, i.e. it can be used to a large part for treating subcases. First, let us point out the following: we have given a characterization for a testing-based preorder for all testable nets; but it is not a priori clear that this result still holds if we restrict the class of nets under consideration – e.g. to the restricted setting where each interface place is *either* an input place with no arc entering *or* an output place with no arc leaving. The reason is that the test nets we used in the proof have IO-places and do not belong to the restricted class.

Luckily, they almost belong to the restricted class: if e.g. a p^+-transition occurs in U then p is an input place of the testable nets, which do not have arcs to p, and there will be no p^--, p^0- or p^1-transitions; hence, we can make p an output place of U with no arcs leaving p. Therefore, our result also holds in this type-restricted case.

Also, the proof can be adapted if we only consider nets without 0-transitions (and without 1-transitions with empty preset): in this case, we would not allow p^0's in refusal sets when defining pr-sequences and -traces; thus, U would have no 0-transitions – the only exception being t_2^4 in the above example-U; this represents a special treatment of the last refusal set, and this is only necessary since we might have time stops. Since these cannot occur in this second restricted class, also t_2^4 can be avoided (details omitted) and the result carries over.

Lastly, U is safe on its non-interface places. (s_2^4 is an exception, if the last refusal set has more than 1 element; but in this case, we could multiply s_2^4 and t_2^4 suitably.) Thus, our construction should also be an essential step for the treatment of safe nets, but here additional work needs to be done; see Section 6.

An important question is now how we show that one net is faster than another. If nets N^{env} were bounded, there would also be only finitely many reachable timed markings; the above characterization would allow to decide $\sqsupseteq$ by a static check plus a decision of regular-language-inclusion. Since N^{env} is generally unbounded, it does not seem feasible to decide $\sqsupseteq$. Still, one can treat many examples (even possibly infinite classes of examples) using simulations:

Definition 4.6 For nets N_1 and N_2, a relation $\mathcal{S}$ between some timed markings of N_1^{env} and some of N_2^{env} is a *(forward) simulation* from N_1 to N_2 if the following hold:

1. $(TM_{N_1}, TM_{N_2}) \in \mathcal{S}$
2. If $(TM_1, TM_2) \in \mathcal{S}$ and $TM_1[t\rangle_{pr} TM'_1$ or $TM_1[X\rangle_{pr} TM'_1$, then for some TM'_2 with $(TM'_1, TM'_2) \in \mathcal{S}$ we have $TM_2[\hat{t}\rangle\rangle_{pr} TM'_2$ or $TM_2[X\rangle\rangle_{pr} TM'_2$, where $\hat{t}$ is t for $t \in T_2^{env} \setminus T_2$ and the empty word λ otherwise. Observe that these moves from TM_2 to TM'_2 may involve several transitions of N_2.

□

The following theorem is straightforward; compare e.g. [LV95] for a similar result and a survey on the use of simulations; note that a simulation does not have to exist in each case where $N_1 \sqsupseteq N_2$.

Theorem 4.7 *If there exists a simulation from N_1 to N_2, then $N_1 \sqsupseteq N_2$.*

We will show some applications of simulations in the next section. We finish this section with:

Theorem 4.8 *$\sqsupseteq$ is a precongruence w.r.t. relabelling and in- and output hiding.*

Proof: Clearly, the static requirements of 4.5 on the interface places are preserved. Additionally, the PR-semantics of $N/^I p$ consists of the pr-traces of N that do not contain p^+; that of $N/^O p$ consists of the pr-traces of N that do not contain p^- or – in the refusal sets – p^1 or p^0; $PR(N[p \to p'])$ can be obtained by replacing all p^+ etc. by p'^+ etc. in the pr-traces of N. □

5 Examples

Our first two examples are more technical in nature; the first discusses aspects of asynchronous communication and timing. N_1 and N_2 in Figure 5 both have the pr-trace $p_2^- p_1^-$, showing that with asynchronous communication messages can be taken 'in the wrong order'.

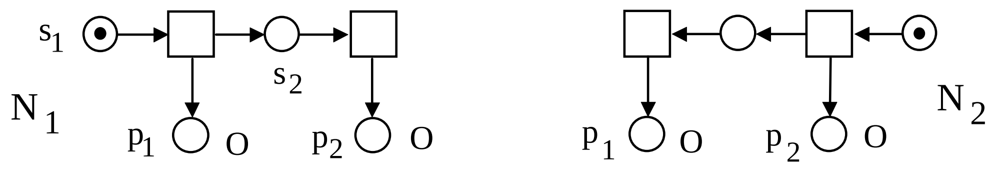

Figure 5

In fact, $N_1 \sqsupseteq N_2$ (and vice versa), since the outputs must appear in the first round. The formal proof is maybe not that obvious: a suitable simulation relates reachable timed markings TM_1 of N_1 and TM_2 of N_2, if they are the initial timed markings or if : $M_1(s_1) = 0$, $M_1(p_1) = M_2(p_1)$, $M_1(p_2) + M_1(s_2) = M_2(p_2)$, the old tokens are the same on the interface places and M_2 is 0 on the other places. When N_1 fires its first transition, N_2 fires both its transitions, giving a related timed marking; then both can perform p_1^-, and before or after N_1 fires its second transition, bringing both nets to essentially the same timed marking. Only then refusal sets can occur.

If we change all transitions to 1-transitions (getting N_1' and N_2'), there could be enough time between appearance of the tokens to observe their order: $p_1^-\{p_2^0\} \in PR(N_1') \setminus PR(N_2')$.

The second example demonstrates the effect of interface typing. Also for N_1 and N_2 in Figure 6, we have $N_1 \sqsupseteq N_2$ and vice versa. (The dead transition in N_1 ensures that the nets satisfy the static condition of Theorem 4.5.) In N_1, tokens on s and on p_2 behave the same: they can get old, and then at the latest they move to the output. Thus, a simulation relates timed markings, if the sum of (old) tokens on s and p_2 in N_1 is the number of (old) tokens on p_2 in N_2, and they are equal on the other places. The situation would be completely different, if p_2 were an IO-place; in this case N_2 would have additional functionality as witnessed by $p_1^+p_2^- \in PR(N_2) \setminus PR(N_1)$.

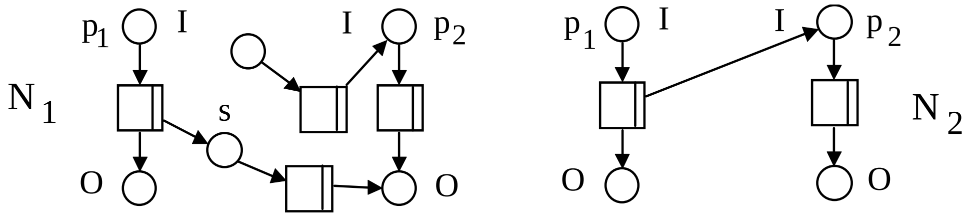

Figure 6

Now we will discuss a series of examples involving the production lines from Section 2. Generally, assume that N_1 has a transition t with a dot, i.e. with a 'private' loop place l, and we split t into two transitions as shown in Figure 7; i.e. t_1 inherits l and the preset of t, and t_2 the postset of t except l.

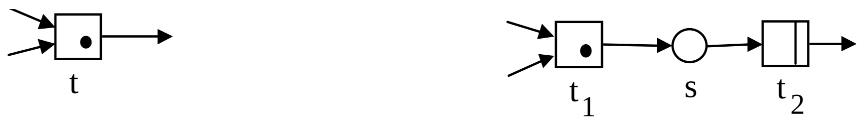

Figure 7

Then, $N_1 \sqsupseteq N_2$: this can be shown with a simulation that relates timed markings TM_1 of N_1 and TM_2 of N_2, if TM_1 is componentwise the restriction of TM_2 to S_1 and $M_2(s) = 0$; N_2 simulates each firing of t by firing t_1t_2. As an application, we have *PL1* $\sqsupseteq$ *PL2* and also e.g. *PL12* := *PL1* $|||$ *PL2* $\sqsupseteq$ *PL2* $|||$ *PL2* =: *PL22* by 3.3. In fact, *PL1* is strictly faster than *PL2*, as witnessed by $p_1^+\emptyset\{p_2^0\} \in PR(PL2) \setminus PR(PL1)$, which means that only with *PL1* an order will certainly result in a product before time 2.

We will now show that *PL22* $\sqsupseteq$ *PL2*, denoting the loop place of t_1 by l and giving a dash to the items of the second copy of *PL2* in *PL22*. A respective simulation relates TM_{22} and TM_2 if they are equal on the interface, $M_{22}(s) + M_{22}(s') =$

$M_2(s)$, ${M^{old}}_{22}(s) + {M^{old}}_{22}(s') = {M^{old}}_2(s)$, $M_{22}(l) = M_{22}(l') = 1 = M_2(l)$ and $\min({M^{old}}_{22}(l), {M^{old}}_{22}(l')) = {M^{old}}_2(l)$.

It is clear that e.g. t_1 or t_1' in *PL22* is simulated by t_1 in *PL2*, and that, after firing them, one of l and l' in *PL22* does not have an old token, which also holds for l in *PL2*, and this is okay since the minimum above is 0. A bit tricky is the occurrence of a refusal set in *PL22*, where the interface transitions are no problem since the interface places are marked the same way in both nets. Let us just consider t_1; since t_1 and t_1' in *PL22* are not enabled by old tokens, p_1 does not carry old tokens, or otherwise at least one of l and l' does not carry old tokens; in either case, p_1 or l does not carry old tokens in *PL2*, which therefore can perform a refusal set as far as t_1 is concerned.

Again, *PL22* is strictly faster – as witnessed by $p_1^+ p_1^+ \emptyset\emptyset p_2^- \{p_2^0\}$, where the tokens on p_1 are old after the first $\emptyset$, but all of them have to leave p_1 before the second $\emptyset$ only in *PL22*.

We close by comparing *PL12* and *PL1*. One might expect the former to be faster, since the additional production line should help somehow; but this is in fact not true: an order might be processed by this slower line giving rise to the slower behaviour $p_1^+ \emptyset \{p_2^0\} \in PR(PL12) \setminus PR(PL1)$.

On the other hand, *PL2* does help indeed if there is a rush of orders. If *PL1* gets four orders, the tokens will be old in the second round; so in this and the third and fourth round, at least one token will be moved to p_2. If these three tokens are removed, the fourth refusal set occurs at a time when p_2 is empty; hence, $p_1^+ p_1^+ p_1^+ p_1^+ \emptyset\emptyset\emptyset p_2^- p_2^- p_2^- \{p_2^0\} \in PR(PL1)$. But this pr-trace is not in $PR(PL12)$: here, in the second round at least two tokens will be moved from p_1 and the remaining ones in the third; thus, in the fourth round there will be some tokens on p_2 and some old tokens on s, which have to be moved; after taking three tokens, p_2 is still not empty at the fourth refusal set. We conclude that *PL12* and *PL1* are incomparable.

Since all phenomena encountered in the above examples had an intuitive explanation, these examples demonstrate that our efficiency preorder makes sense. It should also have become clear, that statements about $\sqsupseteq$ as presented above can only be treated using some characterization – as we have provided it.

6 Conclusion and Work in Progress

For nets as models of asynchronous systems, we have presented a parallel composition $|||$ that merges places and have introduced a typing of such interface places. This composition was used for a testing scenario which gives rise to a faster-than relation for components that communicate asynchronously. (This is in analogy with previous studies for the case of synchronous communication.) We have characterized this faster-than relation and used the characterization to compare the efficiency of some examples.

The most important next step will be to adapt the presented approach to the study of safe nets, and in Augsburg work on this is in progress along the following lines. Since safe nets can be build from components that are not safe in other environments, one possibility is to require that N_1 is a faster implementation of N_2 only if $N_1 \,|||\, U$ is safe whenever $N_2 \,|||\, U$ is; this would lead to considering an additional set of sequences that make a component unsafe – compare $U(N)$ in [Vog92, Section 3.3.1].

Another possibility is to enforce safety on the interface by coupling each interface place with a complement place, and to change the definition of $|||$ accordingly. Then, it is much more reasonable to require the components to be safe in all environments, which makes the *PR*-semantics of a net the language of a finite automaton, so inclusion would certainly be decidable. In fact, wrapping a place-bordered net N into a standard environment in N^{env} wraps the place-oriented problem into a transition-oriented one; thus, it should be possible to check *PR*-inclusion with our tool FASTASY, which was designed for the case of synchronous communication; see [BV98] for results regarding the MUTEX-problem obtained with this tool.

We have convinced ourselves that 0-transitions allow to express solutions to the MUTEX-problem in the cases of asynchronous and synchronous communication. This is remarkable, since [Vog97] relates weakly fair behaviour of ordinary safe nets (which disregards any timing) to timed behaviour of nets with only 1-transitions and shows that in any case the MUTEX-problem cannot be solved – unless one extends nets e.g. with so-called read arcs. We will study how read arcs are related to 0-transitions in general, prove formally how the MUTEX-problem can be solved with 0-transitions and compare in particular MUTEX-solutions in a setting with asynchronous communication.

References

[BDE93] E. Best, R. Devillers, and J. Esparza. General Refinement and Recursion for the Box Calculus. In P. Enjalbert, A. Finkel, and K. W. Wagner, editors, *STACS'93*, Lect. Notes Comp. Sci. 665, 130–140. Springer, 1993.

[Bih98] E. Bihler. Effizienzvergleich bei verteilten Systemen – eine Theorie und ein Werkzeug. Diplomarbeit an der Uni. Augsburg, 1998.

[BV98] E. Bihler and W. Vogler. Efficiency of token-passing MUTEX-solutions – some experiments. In J. Desel et al., editors, *Applications and Theory of Petri Nets 1998*, Lect. Notes Comp. Sci. 1420, 185–204. Springer, 1998.

[Che91] G. Chehaibar. Replacement of open interface subnets and stable state transformation equivalence. In *Proc. 12th Int. Conf. Applications and Theory of Petri Nets, Gjern*, 390–409, 1991.

[dBKPR91] F.S. de Boer, J.N. Kok, C. Palamidessi, and J.J.M.M. Rutten. The failure of failures in a paradigm for asynchronous communication. In J.C.M. Baeten and J.F. Groote, editors, *CONCUR '91*, Lect. Notes Comp. Sci. 527, 111–126. Springer, 1991.

[dBZ99] F.S. de Boer and G. Zavaratto. Generic process algebras for asynchronous communication. In J.C.M. Baeten and S. Mauw, editors, *CONCUR '99*, Lect. Notes Comp. Sci. 1664, 226–241. Springer, 1999.

[DNH84] R. De Nicola and M.C.B. Hennessy. Testing equivalence for processes. *Theoret. Comput. Sci.*, 34:83–133, 1984.

[Gom96] D. Gomm. *Modellierung und Analyse verzögerungs-unabhängiger Schaltungen mit Petrinetzen.* PhD thesis, Techn. Univ. München, Dieter Bertz Verlag, 1996.

[Kin97] E. Kindler. A compositional partial order semantics for Petri net components. In P. Azema et al., editors, *Applications and Theory of Petri Nets 1997*, Lect. Notes Comp. Sci. 1248, 235–252. Springer, 1997.

[LV95] N. Lynch and F. Vaandrager. Forward and backward simulations I: Untimed systems. *Information and Computation*, 121:214–233, 1995.

[Lyn96] N. Lynch. *Distributed Algorithms.* Morgan Kaufmann Publishers, San Francisco, 1996.

[PF77] G. Peterson and M. Fischer. Economical solutions for the critical section problem in a distributed system. In *9th ACM Symp. Theory of Computing*, pages 91–97, 1977.

[Pop91] L. Popova. On time Petri nets. *J. Inform. Process. Cybern. EIK*, 27:227–244, 1991.

[SDdSS94] P. Senac, M. Diaz, and P. de Saqui-Sannes. Toward a formal specification of multimedia synchronization scenarios. *Ann. of telecommunications*, 49:297–314, 1994.

[Val94] A. Valmari. Compositional Analysis with Place-Ordered Subnets. In R. Valette, editor, *Application and Theory of Petri Nets '94*, Lect. Notes Comp. Sci. 815, 531–547. Springer, 1994.

[Vog92] W. Vogler. *Modular Construction and Partial Order Semantics of Petri Nets.* Lect. Notes Comp. Sci. 625. Springer, 1992.

[Vog95] W. Vogler. Faster asynchronous systems. In I. Lee and S. Smolka, editors, *CONCUR 95*, Lect. Notes Comp. Sci. 962, 299–312. Springer, 1995. Full version as Report Nr. 317, Inst. f. Mathematik, Univ. Augsburg, 1995.

[Vog97] W. Vogler. Efficiency of asynchronous systems and read arcs in Petri nets. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *ICALP 97*, Lect. Notes Comp. Sci. 1256, 538–548. Springer, 1997.