Integrating Distributed Algorithms into Distributed Systems

Ekkart Kindler and Sibylle Peuker*

Humboldt-Universität zu Berlin
Institut für Informatik
D-10099 Berlin
Germany

Abstract. Distributed algorithms are often part of a larger distributed system. Usually, the properties of the algorithm are proven for the algorithm in isolation. Then, it is not obvious how the algorithm behaves integrated into a larger system.
In this paper we exploit the simple observation that some actions of a distributed algorithm do not belong to the algorithm but are triggered by the environment. If these actions are distinguished and adequately considered in the verification of the algorithm, basically all properties are still valid for the algorithm as a part of a larger distributed system. This result will be formalized in the setting of the *Distributed Algorithms' Working Notation (DAWN)*.

Keywords: Distributed Algorithms; Petri Nets; Compositionality; Temporal Logic.

1 Introduction

Distributed algorithms solve abstract versions of significant problems occurring in practical distributed computing (cf. [15]). The correctness of an algorithm is usually proven for the algorithm running in isolation. For practical use, however, a distributed algorithm is embedded into a larger system. It is not obvious that the correctness of the algorithm is preserved by the embedding.

Of course, there are techniques which allow to argue about properties of an algorithm which is integrated into some environment. In particular, the *rely/guarantee paradigm* [18, 10] specifies properties of an algorithm depending on properties of the environment. In practice, however, the application of these techniques may become quite complicated. In this paper we will present a simpler method which allows to deal with frequently occurring simpler cases.

If we take a closer look at embeddings of algorithms into systems, we observe that in many cases only some actions of the algorithm are triggered by the environment and all other actions run independently of the environment. If we distinguish actions triggered by the environment and appropriately consider

* supported by the DFG: Kompositionale Verifikation

them in the verification of the algorithm, we can avoid sophisticated verification techniques. For example, in a mutual exclusion algorithm such an action is the decision of some agent to request access to the critical section. If this algorithm is embedded into an environment which explicitly models how this decision is made, the important properties of the mutual exclusion algorithm are preserved.

In this paper, we formalize this observation in the setting of the *Distributed Algorithms' Working Notation* (DAWN) [12, 27, 6, 24]. In DAWN a distributed algorithm is modeled by an algebraic Petri net [21, 11]. The concept for integrating an algorithm into an environment are *external transitions* [22, 13]. We justify the concept of external transitions by the following result: Basically, all linear-time properties of a distributed algorithm are preserved, if the distributed algorithm is only synchronized with the environment at external transitions. It is well-known that safety properties are preserved, if transitions of a Petri net are synchronized with the transitions of another Petri net (e.g. [20, 19]). The preservation of liveness properties, however, is more involved and needs different arguments.

We prove this result for an interleaving semantic. Though the result seems to be obvious we show that it does not hold for a partial order semantic. The reason is that additional synchronization at external transitions may restrict concurrency in the algorithm.

We will present this result for low-level Petri nets[1] only, though it can be canonically extended to algebraic high-level Petri nets. This helps to focus on the basic idea rather than on technical details.

The paper is organized as follows: In Sect. 2 we illustrate the main idea of this paper by means of an example. In Sect. 3 we formalize the basic concepts, which will be used in Sect. 4 for formalizing and verifying the main result. In Sect. 5 we show a simple application of this result. In Sect. 6 we show the impact of the result on DAWN.

2 An example

In this section, we will illustrate the main idea of external transitions by a simple example. The Petri net of Fig. 1 models the following algorithm: When triggered by the environment, it searches for some information; then it sends the information to the buffer of a printer, which prints the information.

Technically the possible states of the algorithm are modeled by *places*, which are graphically represented by circles. Initially, the algorithm is idle, the buffer is empty, and the printer is ready. The possible state changes are modeled by *transitions*, which are graphically represented by squares. For example, transition t1 models a state change, which is triggered by the environment: It changes the state from idle to searching. All other transitions are executed by the algorithm independent of the environment. For example, whenever the printer is printing, transition t4 will eventually occur and change the state from printing to ready

[1] We use a version of P/T-systems in this paper which we call *system nets*.

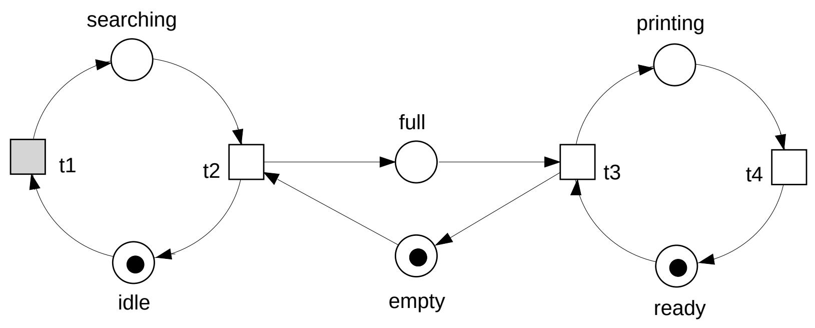

Fig. 1. The system net Σ_1: An agent providing information

again. This is called the *progress assumption* and the corresponding transitions are called *progress transitions*.

The algorithm need not change its state from idle to searching, since the decision for searching for some information is up to the environment. This means that the progress assumption does not apply for transition t1. In order to make this difference explicit, we call t1 an external transition and represent it by a shaded square.

The algorithm satisfies the following property: It will always reach the state idle again. This can be expressed in linear-time temporal logic by the formula $\neg\mathsf{idle} \rhd \mathsf{idle}$. Furthermore, whenever the algorithm is searching the printer will eventually be printing. This can be expressed by the formula $\mathsf{searching} \rhd \mathsf{printing}$.

The above properties can be easily proven in DAWN. Next, we will show how these properties can be exploited if the algorithm is a part of a larger system without considering the algorithm again. To this end, we embed the algorithm into an environment as shown in Fig. 2.

The algorithm's environment consists of an agent which is working initially. If it requires some information it sends a request and becomes waiting. It is waiting until the formerly described algorithm provides the information. Then, it becomes working again. Transition t1 is not an external transition anymore because in system Σ_2 its environment is explicitly modeled.

In this extended system, the only synchronization imposed on the algorithm from Fig. 1 is at transition t1 (t1 may occur only if there is a request). Since t1 is an external transition of the algorithm and therefore we did not assume progress of this transition, all properties of the algorithm are still valid in the system Σ_2. In particular, $\neg\mathsf{idle} \rhd \mathsf{idle}$ and $\mathsf{searching} \rhd \mathsf{printing}$ are still valid. We can use them for Σ_2 without proving them again. In Sect. 5 we will show that it is easy to prove that every request leads to a printed information in the system Σ_2 by exploiting these properties and investigating the interface transitions t1 and t4.

We call an embedding which imposes only synchronizations on external transitions of the algorithm a *conservative extension*. This notion will be justified by

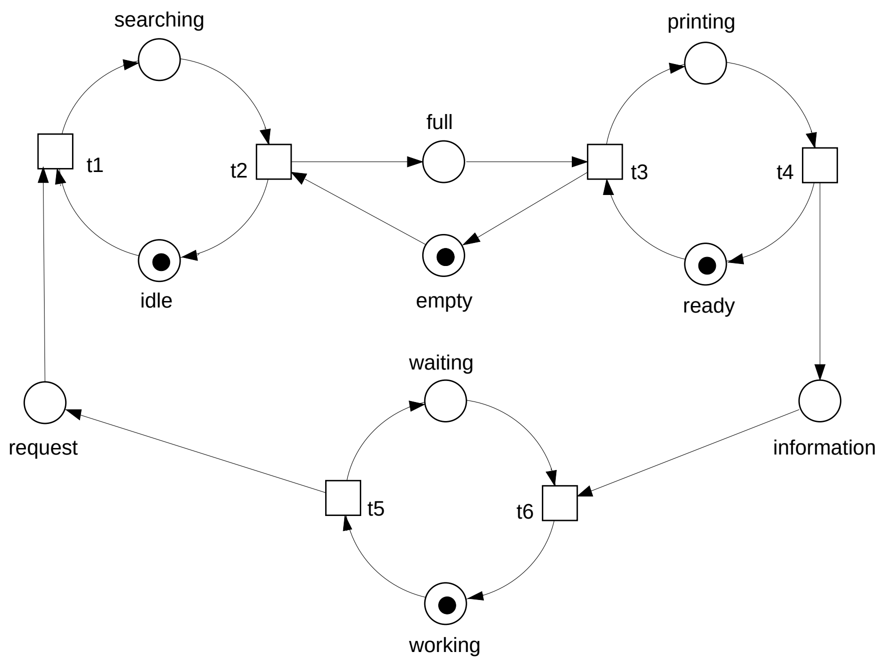

Fig. 2. The system net Σ_2: A conservative extension of Σ_1

the forthcoming Theorem 1. It will be shown that all stuttering invariant properties (cf. [1]) of the algorithm still hold in a conservative extension. Stuttering invariance of a property intuitively means that the validity of the property in a system can not be invalidated by adding a completely independent part to the system. For instance, it can be shown that all properties expressible in linear-time temporal logic [16] without the next-step-operator are stuttering invariant and in [2] is stated that every TLA formula is stuttering invariant.

3 Preliminaries

In this section we formalize the basic concepts: We define Place/Transition Petri nets (P/T-nets) without capacities and arc weights, and system nets (P/T-nets with an initial marking). Furthermore, we formalize conservative extensions, and stuttering invariant properties of a system net.

3.1 Nets and steps

For a smooth presentation of our results we fix a universal set of places $\mathcal{P}$ for this paper. Every net considered in this paper has only places chosen from this set.

Definition 1. (Net, marking) *A* net *is a triple $N = (P, T, F)$ where $P \subseteq \mathcal{P}$ and T are finite disjoint sets and F is a subset of $(T \times P) \cup (P \times T)$. A* universal marking *is a function $M : \mathcal{P} \longrightarrow \mathbb{N}$. A* marking of the net N *is a universal marking if for every $p \in \mathcal{P} \setminus P$ holds $M(p) = 0$.*

The sets P, T and F are called set of *places*, set of *transitions*, and set of *arcs* (or *flow relation*) of the net, respectively. For convenience we will adopt the following convention: An index of a net implicitly transfers to its constituents; for example N_1 has the set of places P_1, the set of transitions T_1 and the flow relation F_1. A marking represents a global state of a net. It assigns to every place the number of tokens carried in that state.

Definition 2. (Preset, postset, step) *Let $N = (P, T, F)$ be a net. The* preset *and the* postset *of an element $x \in P \cup T$ are defined by*

$$\begin{aligned} \mathrm{pre}_N(x) &= \{y \in P \cup T \mid (y, x) \in F\} \\ \mathrm{post}_N(x) &= \{y \in P \cup T \mid (x, y) \in F\} \end{aligned}$$

A transition $t \in T$ is enabled *at a marking M of N, if for all $p \in \mathrm{pre}_N(t)$ holds $M(p) > 0$. If t is enabled at M, it may* occur *and the resulting marking M' is given by*

$$\begin{aligned} M'(p) &= M(p) - 1 && \textit{for all } p \in \mathrm{pre}_N(t) \setminus \mathrm{post}_N(t), \\ M'(p) &= M(p) + 1 && \textit{for all } p \in \mathrm{post}_N(t) \setminus \mathrm{pre}_N(t), \\ M'(p) &= M(p) && \textit{for all other } p \in \mathcal{P}. \end{aligned}$$

The ocurrence of a transition $\mathtt{t}$ *at M resulting in the marking M' we call a* step *and denote it by $M \xrightarrow{t} M'$.*

3.2 System nets and runs

For modeling a system, we add an initial state and some liveness assumptions. For transitions which are triggered by the environment, it depends on the environment whether a transition occurs or not. Therefore, we distinguish two kinds of transitions: *progress transitions* and *external transitions*. There are no liveness assumptions for external transitions. In the graphical representation of a net an external transition is shaded. For an enabled progress transition we assume that either itself or a conflicting transition will eventually occur; where two transitions are in conflict if their presets are not disjoint.

Definition 3. (System net) *A* system net *(N, M, L) is a net N together with an initial marking M of N and a set $L \subseteq T$ of progress transitions.*

From now on, we only consider nets N for which $\mathrm{pre}_N(t) \neq \mathrm{post}_N(t)$ for all $\mathtt{t} \in T$. This restriction is not necessary but it allows us to define runs of a system net as certain sequences of markings without a representation of the occurring transitions. It will become important later that each occurring transition in fact changes the marking of the net and therefore, a stuttering step may be recognized as not belonging to the run of a system net.

Definition 4. (Run of a system net) *A* universal run *is a (finite or infinite) sequence of universal markings. The set of all universal runs is denoted by $\mathcal{M}^\infty$. The* length $|\sigma|$ *of a run σ is the number of sequence elements; it is a natural number if σ is finite, and it is ω if σ is infinite.*

Let $\Sigma = (N, M, L)$ be a system net. A run of Σ *is a (finite or infinite) sequence $\rho = \langle M^0, M^1, \ldots \rangle$ of markings of N such that*

(i) $M^0 = M$,
(ii) for every marking M^i of the sequence with $i > 0$ there is a transition $t \in T$ such that $M^{i-1} \xrightarrow{t} M^i$ is a step, and
(iii) for every progress transition $t \in L$ holds: If t is enabled at marking M^i of the sequence then there is a $j \in \mathbb{N}$ and a transition $t' \in T$ such that $i < j < |\rho|$, $\mathrm{pre}_N(t) \cap \mathrm{pre}_N(t') \neq \emptyset$ and $M^{j-1} \xrightarrow{t'} M^j$ is a step.

The set of all runs of Σ is denoted by $\mathcal{R}(\Sigma)$.

Condition (iii) formalizes the progress assumption: an enabled progress transition either will occur itself or a conflicting transition will occur. With this assumption the runs of a system net are exactly the sequential runs which we obtain by sequentializing the non-sequential runs of DAWN [27] (see also Sect. 6).

Let M be a universal marking and let P be a subset of $\mathcal{P}$. The *restriction* of M to P is the function $M|_P : \mathcal{P} \longrightarrow \mathbb{N}$, s.t. $M|_P(p) = M(p)$ for all $p \in P$ and $M|_P(p) = 0$ for all $p \in \mathcal{P} \setminus P$. The restriction $\rho|_P$ of a universal run $\rho = \langle M^0, M^1, \ldots \rangle$ is the sequence $\langle M^0|_P, M^1|_P, \ldots \rangle$ of restricted markings.

3.3 Conservative extensions of system nets

Now, we will define conservative extensions of a system net, which reflect the correct embedding of an algorithm into a system. As stated before, external transitions of an algorithm are triggered by the environment. Therefore, in a conservative extension we allow additional synchronizations for external transitions (input arcs from places of the environment). Moreover, we allow additional output arcs at all transitions of the algorithm to places of the environment, because output arcs do not affect the behavior of the algorithm.

Definition 5. (Conservative extension) *Let $\Sigma_1 = (N_1, M_1, L_1)$ and $\Sigma_2 = (N_2, M_2, L_2)$ be two system nets. Σ_2 is a* conservative extension *of Σ_1 if the following four conditions hold:*

(i) $P_1 \subseteq P_2$, $T_1 \subseteq T_2$ and $L_1 \subseteq L_2$.
(ii) For every place $p \in P_1$ holds

$$\mathrm{pre}_{N_1}(p) = \mathrm{pre}_{N_2}(p) \quad \textit{and} \quad \mathrm{post}_{N_1}(p) = \mathrm{post}_{N_2}(p).$$

(iii) For every $t \in L_1$ holds

$$\text{pre}_{N_1}(t) = \text{pre}_{N_2}(t).$$

(iv) $M_1 = M_2|_{P_1}$.

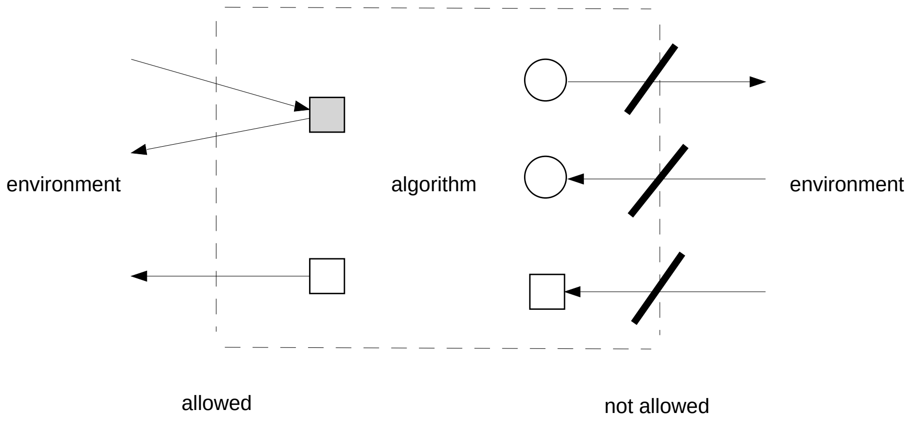

Fig. 3. An illustration for Def. 5

Figure 3 illustrates condition (ii) and (iii) of the above definition. The left side shows the allowed cases and the right side shows the cases not allowed. The "new parts" of the net (the environment) are connected with the "old parts" (the algorithm) only via input or output arcs of external transitions of the algorithm or output arcs of progress transitions. By condition (iv) the initial marking of the places of P_1 is the same in both nets.

3.4 Properties and stuttering invariance

In the literature properties of a system may be syntactically expressed in many different ways; for instance as a formula of temporal logic. Our result, however, does not depend on a particular syntactical representation. Therefore, we do not fix a syntactical representation but deal with properties semantically (as a subset of all possible behaviors of the system).

Definition 6. (Property) *A* property *H is a set of universal runs, i.e. $H \subseteq \mathcal{M}^\infty$. A system net Σ satisfies H (denoted by $\Sigma \models H$) if $\mathcal{R}(\Sigma) \subseteq H$.*

Let $N = (P, T, F)$ be a net. We say that H is a property over *N if H is a property and if for every two universal runs ρ and σ holds: If $\rho \in H$ and $\rho|_P = \sigma|_P$, then $\sigma \in H$.*

Intuitively, a property over N is a property for which only the marking of the places of P matters whereas the marking of the places $\mathcal{P} \setminus P$ is irrelevant.

For defining stuttering invariance we introduce some notions about sequences (cf. [1]). Let $\sigma = \langle x_0, x_1, \ldots \rangle$ be a sequence. The sequence σ is *stutter-free* if for all $i < |\sigma| - 1$ holds $x_i \neq x_{i+1}$. The *stutter-free version* $\natural\sigma$ of σ is obtained by removing all stuttering steps from σ. Technically, the stutter-free version of σ is defined as follows: First, we inductively define a sequence of pairs $\sigma' = \langle (y_0, z_0), (y_1, z_1), \ldots \rangle$:

(i) $(y_0, z_0) = (x_0, 0)$.

(ii) Let (y_i, z_i) be already defined. If there exists a natural number

$$k = \min\{m > z_i \mid y_i \neq x_m\}$$

then we define $(y_{i+1}, z_{i+1}) = (x_k, k)$. Otherwise (y_{i+1}, z_{i+1}) is undefined and $|\sigma'| = i + 1$.

Now $\natural\sigma = \langle y_0, y_1, \ldots \rangle$ is the sequence of the first elements of the pairs of σ'.

For example, $\natural\langle x_0, x_0, x_1, x_2, x_2, x_2 \rangle = \langle x_0, x_1, x_2 \rangle$ and $\natural\langle x, x, x, \ldots \rangle = \langle x \rangle$. Note that the infinite stuttering step of the second example reduces to a sequence of length 1.

Definition 7. (Stuttering invariant property) *A property H is* stuttering invariant *(or closed under stuttering) if for every two universal runs ρ and σ holds: If $\rho \in H$ and $\natural\rho = \natural\sigma$, then $\sigma \in H$.*

4 Preservation of stuttering invariant properties

In this section, we will prove that all stuttering invariant properties of a system still hold in each conservative extension. The main argument is that every run of the extension can be restricted to a run of the initial system net possibly extended by stuttering steps. The reason is that every progress transition of the initial system net may occur independently of the state of the environment (the "new parts") of the net. This result is given in Prop. 1. The proof is straightforward.

Proposition 1. *Let Σ_1 and Σ_2 be two system nets such that Σ_2 is a conservative extension of Σ_1. If ρ is a run of Σ_2 then $\natural(\rho|_{P_1})$ is a run of Σ_1.*

Proof: Let $\rho = \langle M^0, M^1, \ldots \rangle$ be a run of Σ_2. By definition the restriction $\rho|_{P_1}$ is a sequence of markings of N_1. Then, by definition of the stutter-free version also $\natural(\rho|_{P_1}) = \langle K^0, K^1, \ldots \rangle$ is a sequence of markings of N_1. To show that $\natural(\rho|_{P_1})$ is a run of Σ_1 we have to prove that it satisfies conditions (i)–(iii) of Def. 4.

(i) First we show that $K^0 = M_1$: By definition of $\natural(\rho|_{P_1})$ we have that $K^0 = M^0|_{P_1}$. By definition of the conservative extension we have $M_1 = M_2|_{P_1}$. By Def. 4 we have $M^0 = M_2$, which implies $M^0|_{P_1} = M_2|_{P_1}$. Altogether, we have $K^0 = M^0|_{P_1} = M_1$. Therefore, condition (i) is satisfied.

(ii) Now we show that $\natural(\rho|_{P_1})$ is a sequence of steps of Σ_1: Because ρ is a run of Σ_2, for every $i < |\rho| - 1$ there exists a transition $t \in T_2$ such that $M^i \xrightarrow{t} M^{i+1}$ is a step in N_2. If $t \in T_2 \setminus T_1$ then by (ii) of Def. 5

$$\big(\mathrm{pre}_{N_2}(t) \cup \mathrm{post}_{N_2}(t)\big) \cap P_1 = \emptyset,$$

hence, the marking of the places of P_1 remains unchanged after occurrence of t. In this case $M^i|_{P_1} = M^{i+1}|_{P_1}$.
If $t \in T_1$ then by (ii) of Def. 5

$$\mathrm{pre}_{N_1}(t) \subseteq \mathrm{pre}_{N_2}(t) \text{ and } \mathrm{post}_{N_1}(t) \subseteq \mathrm{post}_{N_2}(t),$$

hence, t is also enabled in the marking $M^i|_{P_1}$ of the net N_1 and the resulting marking after firing of t in N_1 is $M^{i+1}|_{P_1}$.
Because of these facts $\rho|_{P_1} = \langle M^0|_{P_1}, M^1|_{P_1}, \ldots\rangle$ is a sequence of markings of N_1 such that for every $i < |\rho| - 1$ holds either $M^i|_{P_1} = M^{i+1}|_{P_1}$ or there is a transition $t \in T_1$ such that $M^i|_{P_1} \xrightarrow{t} M^{i+1}|_{P_1}$ is a step in N_1. By removing all stuttering steps of this sequence we get the sequence $\natural(\rho|_{P_1}) = \langle K^0, K^1, \ldots\rangle$ such that $K^i \xrightarrow{t} K^{i+1}$ for some $t \in T_1$. Hence $\natural(\rho|_{P_1})$ satisfies condition (ii) of Def. 4.

(iii) Let $t \in L_1$. We have to show that $\natural(\rho|_{P_1})$ satisfies the progress assumption for t. First we note that Def. 5 implies $t \in L_2$ and

$$\mathrm{pre}_{N_1}(t) = \mathrm{pre}_{N_2}(t). \tag{1}$$

Let t be enabled at some marking K^i of $\natural(\rho|_{P_1})$. We have to show that there is a j with $i < j < |\natural(\rho|_{P_1})|$ such that $K^{j-1} \xrightarrow{t'} K^j$ is a step in N_1 for some $\mathsf{t}' \in T_1$. By definition of $\natural(\rho|_{P_1})$ there exists an $i' \geq i$ such that $M^{i'}|_{P_1} = K^i$. Because of equation (1) t is also enabled for the system net Σ_2 in $M^{i'}$. Because $t \in L_2$ there are $t' \in T_2$ and $j' > i'$ such that

$$\mathrm{pre}_{N_2}(t) \cap \mathrm{pre}_{N_2}(t') \neq \emptyset \tag{2}$$

$$M^{j'-1} \xrightarrow{t'} M^{j'} \tag{3}$$

From Def. 5 (ii) and (iii) t' can only be an element of T_1. This implies together with (3)

$$M^{j'-1}|_{P_1} \neq M^{j'}|_{P_1}.$$

(Here we need the convention that the preset of a transition does not equal its postset.) Hence, $M^{j'-1}|_{P_1}, M^{j'}|_{P_1}$ is no stuttering step and therefore, not removed from the sequence of restricted markings. So, there exists a $j > i$ such that $K^{j-1} = M^{j'-1}|_{P_1}$ and $K^j = M^{j'}|_{P_1}$. Because of the equations (2) and (1) and because $\mathrm{pre}_{N_2}(t) \cap \mathrm{pre}_{N_2}(t') \subseteq P_1$ we obtain

$$\mathrm{pre}_{N_1}(t) \cap \mathrm{pre}_{N_1}(t') \neq \emptyset.$$

Finally, from (3) and $t' \in T_1$ we infer that $K^{j-1} \xrightarrow{t'} K^j$ is a step in Σ_1.
Hence $\natural(\rho|_{P_1})$ satisfies condition (iii) of Def. 4 which completes the proof. □

Now we are able to prove the main result.

Theorem 1. *Let $\Sigma_1 = (N_1, M_1, L_1)$ and $\Sigma_2 = (N_2, M_2, L_2)$ be two system nets such that Σ_2 is a conservative extension of Σ_1. For every stuttering invariant property H over N_1 holds:*

$$\textit{If} \qquad \Sigma_1 \models H \qquad \textit{then} \qquad \Sigma_2 \models H.$$

Proof: Let H be a stuttering invariant property over N_1 such that $\Sigma_1 \models H$. Let ρ be a run of Σ_2. We have to show that $\rho \in H$. From Prop. 1 and the fact that Σ_1 satisfies H we obtain that $\natural(\rho|_{P_1}) \in H$. Because of the stuttering invariance of H we can infer that $\rho|_{P_1} \in H$ and because H is a property over N_1 we finally get $\rho \in H$. Because ρ was an arbitrary run of Σ_2, we have $\Sigma_2 \models H$. $\square$

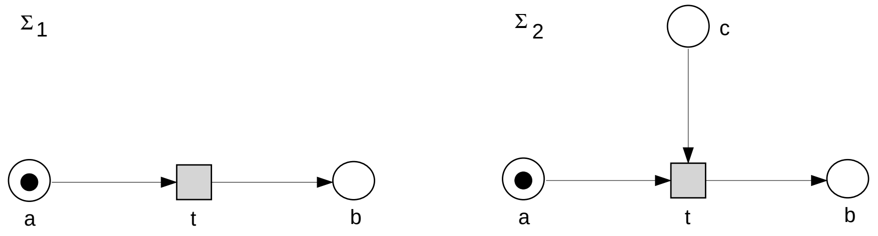

Fig. 4. A counter-example for the reverse direction of Theorem 1

The reverse direction of the implication in Theorem 1 does not hold. Figure 4 shows a counter-example. The system net Σ_2 with the external transition t and the initial marking M is a conservative extension of the system net Σ_1. The only run of Σ_2 is $\langle M \rangle$. In Σ_2 the place a is always marked. The system net Σ_1, however, does not satisfy this property because there is a run $\langle M, M' \rangle$ where $M'(b) = 1$ and $M'(p) = 0$ for every other $p \in \mathcal{P}$; in particular $M'(a) = 0$.

5 The example — continued

In the previous sections, we have shown how to integrate a distributed algorithm into a distributed system such that its properties are preserved. We have formulated the result independent of a particular syntactic representation of properties. Therefore, we are free to use any syntactic representation of properties as long as all properties expressible are stuttering invariant and the properties are only about the algorithm. For example, we could use linear-time temporal logic for this purpose [17] without a next-step operator. The exclusion of the next-step operator guarantees stuttering invariance of the properties. A formula which uses only symbols of the embedded system net, denotes a property which is about the algorithm only.

In this section, we will continue our example from Sect. 2 and will apply Theorem 1 where properties are expressed by temporal formulas of DAWN. DAWN formulas of the form $\Box\,\varphi$ and $\varphi \rhd \bigvee_{i \in I} p_i$ where φ is an arbitrary state-formula over system net Σ and p_i are places of Σ satisfy the properties required for applying Theorem 1. By the *always* operator $\Box$, we are able to express properties of the form: A property φ holds always in every run the system. By the *leadsto* operator $\rhd$, we are able to express properties of the form: If a property φ holds once in a run of the system, another property ψ will eventually hold later in this run. Note that the state-formulas allowed on the right-hand side of the *leadsto* operator are not arbitrary, which will be explained in Sect. 6 in some more detail.

As shown in Sect. 2, system net Σ_1 from Fig. 1 satisfies $\neg\mathsf{idle} \rhd \mathsf{idle}$ and $\mathsf{searching} \rhd \mathsf{printing}$. By Theorem 1 we know that these properties are still valid in Σ_2 shown in Fig. 2. Now, we will show that $\mathsf{request} \rhd \mathsf{information}$ is valid in Σ_2, which means that any request will eventually result in some information. To this end, we will only use the already proven properties of Σ_1 and some arguments about the interface. To make sure that we do not use other information of Σ_1 we represent only the interface and the already proven properties in Fig. 5.

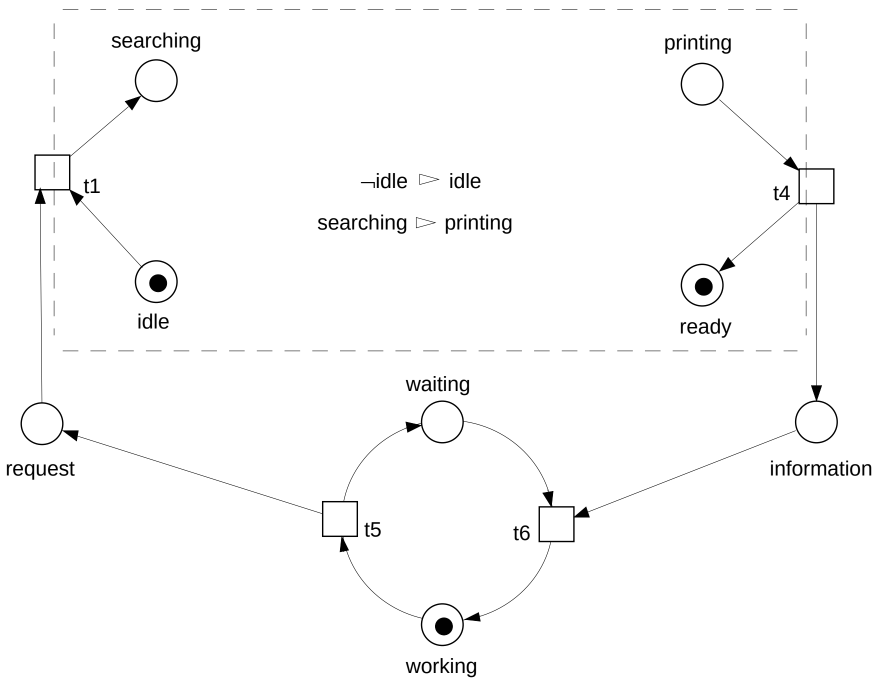

Fig. 5. Properties and interface

From $\neg$idle $\rhd$ idle we know that every request will eventually be accepted; i.e. a request leads to a searching agent (request $\rhd$ searching). This can be easily picked up from the net at the interface transition t1 and is justified by the "progress pick up rules" provided in DAWN (see [27, 23]). Furthermore, we know from Σ_1 that a searching agent always leads to a printing printer: searching $\rhd$ printing. Finally, printing enables the interface transition t4 which will eventually occur and mark information. Altogether, we have shown request $\rhd$ information. Note, that we did not consider the whole system net Σ_1! We only argued on the *interface transitions* t1 and t4 and the already known properties of Σ_1 which are still valid in Σ_2 due to Theorem 1.

In this way, any distributed algorithm which is proven correct may be integrated into some environment and properties of the whole system can be verified without a detailed investigation of the algorithm. It was only necessary to consider the interface and the properties of the algorithm.

6 Concurrency

In the previous section, we used DAWN formulas of a specific form for applying Theorem 1. In particular, we did only apply the Theorem for leadsto formulas of the form $\varphi \rhd \bigvee_{i \in I} p_i$. The reason is that in DAWN [12, 27, 6, 24] formulas are interpreted on non-sequential runs [8, 4]—in contrast to Theorem 1 which is formalized for sequential runs only. For formulas of the above form, both interpretations coincide. Therefore, we can apply Theorem 1.

For arbitrary leadsto-formulas, however, the interpretation on sequential runs and on non-sequential runs does not coincide. Even worse, Theorem 1 does not hold in this general case, since concurrency of an algorithm may be lost when integrated into a system. We will give a simple counter-example below.

We will explain our counter-example without formalizing non-sequential runs and the interpretation of DAWN-formulas (see [13] for formal definitions). Figure 6

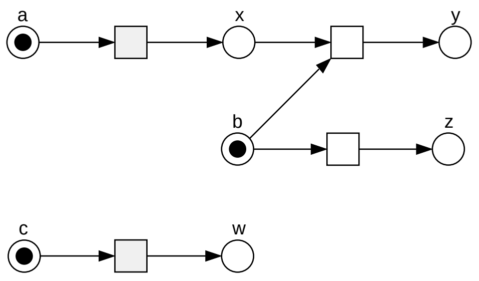

Fig. 6. A system net Σ

shows a simple system net Σ and Fig. 7 shows a non-sequential run of Σ.

The basic concept for defining the validity of temporal formulas is the *cut* of a non-sequential run—which corresponds to a state. A cut is a maximal set of

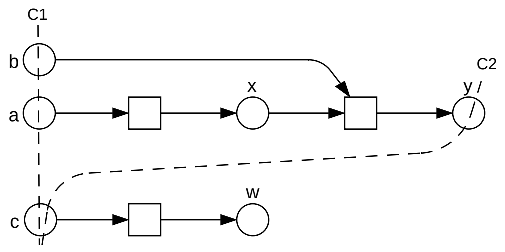

Fig. 7. A run of system net Σ

places which are not ordered by the arcs of the non-sequential run. For example, in the non-sequential run of Fig. 7 we have marked two cuts $C_1 = \{b, a, c\}$ and $C_2 = \{y, c\}$ by dashed lines; the cut C_1 is followed by cut C_2. Then, leadsto property $\varphi \rhd \psi$ is valid in a non-sequential run, if every cut which satisfies φ is followed by a cut satisfying ψ. The property is valid in a system, if every non-sequential run of the system satisfies the property.

Clearly, $(a \wedge b \wedge c) \rhd (z \vee (y \wedge c))$ is valid in the system shown in Fig.6—in each non-sequential run, a cut which contains a, b, and c is followed by a cut which satisfies $(z \vee (y \wedge c))$. Note, that this cut need not be present in each sequential version of this run. Now, consider the conservative extension of Σ to Σ' as shown

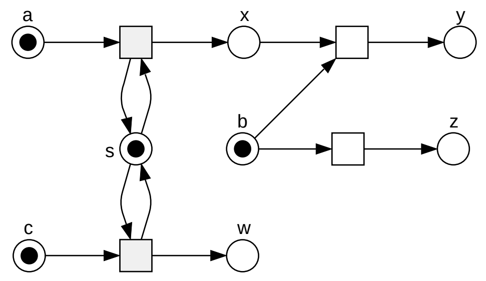

Fig. 8. System net Σ': a conservative extension of Σ

in Fig. 8. Figure 9 shows one non-sequential run of Σ'. In this non-sequential run, there is no cut satisfying $z \vee (y \wedge c)$; therefore, $(a \wedge b \wedge c) \rhd (z \vee (y \wedge c))$ does not hold.

Altogether, this counter-example demonstrates two things:

(i) The validity of Theorem 1 is not completely trivial, since it does only hold for sequential semantics.
(ii) Theorem 1 does not hold for *true-concurrency semantics*. Basically, this means that *concurrency should not be specified* for algorithms which are supposed to be included in a system; this way, we add a technical argument to the arguments of [9].

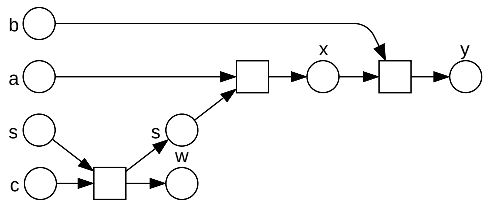

Fig. 9. A non-sequential process of Σ'

For most algorithms, however, this negative result does not do any harm, because these can be easily specified by DAWN-formulas which satisfy the above restrictions [27, 24]. For these algorithms, Theorem 1 can be applied for integrating them into distributed systems.

Still, there are some so-called *round based* [25, 26] algorithms for which concurrency is explicitly specified; these properties may be lost when the algorithms are integrated into an arbitrary system.

7 Conclusion

This paper presents a simple mechanism for integrating distributed algorithms into distributed systems. The employed concept of *external transitions* was inspired by Dijkstra's observation [7] that a *mutual exclusion algorithm* must also work if one participant does not want to enter the critical section any more. So, external transitions have been introduced to DAWN from its very beginning [22, 13, 11] (with different names and different graphical representations). In this paper, we have given a theoretical justification for external transitions.

Of course, other formalisms provide concepts for integrating distributed algorithms (or protocols) into distributed systems. For example, UNITY [5] or TLA [14] use *unless* properties in order to represent possible steps of the environment. In these approaches, the possible behavior of the environment is expressed in terms of temporal logic. There are even more sophisticated techniques which use the so-called *rely/guarantee* paradigm [10, 2, 3] in order to incorporate assumptions on the environment. Though appealing from a theoretical point of view, it is often more convenient to have the assumptions on the environment in the operational model of the distributed algorithm. This boils down to external transitions.

References

1. Martín Abadi and Leslie Lamport. The existence of refinement mappings. *Theoretical Computer Science*, 82:253–284, 1991. previously: SRC Research Report 27, April 1988.
2. Martín Abadi and Leslie Lamport. Conjoining specifications. Research Report 118, Digital Equipment Corporation, System Research Center, December 1993.

3. Martín Abadi and Gordon D. Plotkin. A logical view of composition. *Theoretical Computer Science*, 114:3–30, 1993.
4. Eike Best and César Fernández. *Nonsequential Processes*, volume 13 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1988.
5. K. M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.
6. Jörg Desel and Ekkart Kindler. Proving correctness of distributed algorithms using high-level Petri nets – a case study. In *Proceedings CSD 98*. IEEE Computer Society Press, March 1998. to appear.
7. E. W. Dijkstra. Solution of a problem in concurrent programming control. *Communication of the ACM*, 8(9):569, 1965.
8. U. Goltz and W. Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.
9. Rosalind L. Ibrahim, John A. Ogden, and Shirley A. Williams. Should concurrency be specified? In C. Rattray, editor, *Specification and Verification of Concurrent Systems*, Workshops in Computing, pages 246–271. Springer-Verlag, 1990.
10. Cliff. B Jones. Specification and design of (parallel) programs. In R.E.A Mason, editor, *Information Processing*, pages 321–332. IFIP, Elsevier Science Publishers B.V. (North Holland), 1983.
11. Ekkart Kindler and Wolfgang Reisig. Algebraic system nets for modelling distributed algorithms. *Petri Net Newsletter*, 51:16–31, December 1996.
12. Ekkart Kindler, Wolfgang Reisig, Hagen Völzer, and Rolf Walter. Petri net based verification of distributed algorithms: An example. *Formal Aspects of Computing*, 9:409–424, 1997.
13. Ekkart Kindler and Rolf Walter. Message passing mutex. In J. Desel, editor, *Structures in Concurrency Theory*, Workshops in Computing, pages 205–219, Berlin, May 1995. Springer-Verlag.
14. Leslie Lamport. The temporal logic of actions. SRC Research Report 79, Digital Equipment Corporation, Systems Research Center, December 1991.
15. Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
16. Zohar Manna and Amir Pnueli. Completing the temporal picture. *Theoretical Computer Science*, 83(1):97–130, 1991.
17. Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems — Specification*. Springer-Verlag, 1992.
18. Jayadev Misra and K. Mani Chandy. Proofs of networks of processes. *IEEE Transactions on Software Engineering*, SE-7(4):417–426, July 1981.
19. J. Padberg, M. Gajewsky, and C. Ermel. Rule-based refinement of high-level nets preserving safety properties. 1998. accepted for ETAPS–FASE.
20. Sibylle Peuker. Invariant property preserving extensions of elementary Petri nets. In H. Ehrig, W. Reisig, and H. Weber, editors, *Move-On-Workshop der DFG-Forschergruppe Petrinetz-Technologie*, volume 97-21 of *Forschungsberichte des Fachbereichs Informatik*, pages 143–150. Technische Universität Berlin, April 1997.

21. Wolfgang Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80:1–34, May 1991.
22. Wolfgang Reisig. Petri net models of distributed algorithms. In Jan van Leeuwen, editor, *Computer Science Today: Recent Trends and Developments*, volume 1000 of *LNCS*, pages 441–454. Springer-Verlag, 1995.
23. Wolfgang Reisig. Interleaved progress, concurrent progress and local progress. In Doron A. Peled, Vaughan R. Pratt, and Gerard J. Holzmann, editors, *Workshop on Partial Oder Methods in Verification (Princeton 96)*, volume 29 of *DIMACS*, pages 99–115. American Mathematical Society, 1996.
24. Wolfgang Reisig. *Elements of Distributed Algorithms — Modeling and Analysis with Petri Nets*. Springer-Verlag, 1998.
25. Rolf Walter. Races in rounds. *Petri Net Newsletter*, 48:3–6, April 1995. Cover Picture Story.
26. Rolf Walter. The asynchronous stack revisited: Rounds set the twilight reeling. In C. Freksa, M. Jantzen, and R. Valk, editors, *Foundations of Computer Science: Potential – Theory – Cognition*, volume 1337 of *LNCS*, pages 307–312. Springer-Verlag, 1997.
27. M. Weber, R. Walter, H. Völzer, T. Vesper, W. Reisig, S. Peuker, E. Kindler, J. Freiheit, and J. Desel. DAWN: Petrinetzmodelle zur Verifikation Verteilter Algorithmen. Informatik-Bericht 88, Humboldt-Universität zu Berlin, December 1997.