# A Temporal Logic for Events and States in Petri Nets*

Ekkart Kindler, Tobias Vesper

Humboldt-Universität zu Berlin, Institut für Informatik, Unter den Linden 6, D-10099 Berlin†

**Abstract**

In some phases of system development *state-based* methods are adequate; in others *event-based* methods are adequate. Petri nets provide a system model which supports both methods and thus allow a smooth transition between the different phases of system development. Most temporal logics for Petri nets, however, do not support both methods.

In this paper we introduce a temporal logic for Petri nets which allows to argue on states as well as on events. This way, specifications in the early phases can be event based and verification in later phases can be state-based within a single formalism.

**Keywords**: Temporal logic; event-based; state-based; Petri nets; system development; specification; verification.

## 1 Introduction

Most *formal methods* employed for the specification and development of distributed systems are either *event-based* or *state-based*. The event-based methods highlight the events and their relation; the state-based methods emphasize the states and the state changes. Both approaches have their pros and cons which should not be played off against each other.

For system development both views seem to be equally important [vdA96]. For example, in the case study HDMS of the project KORSO [CHL95] the informal requirements are event-based. In the later phases properties are given state-based and are verified in a state-based temporal logic [EGVW97]. More generally, we believe that in the early phases of systems development event-based methods are often more suitable than state-based methods; in contrast to later phases where often state-based methods seem to be more suitable. The reason is that — in the field of business processes — users describe their business in terms of activities and events which trigger these activities; in order to communicate with the users in the early design phases, event-based description techniques are necessary. In later phases the use of state-based techniques is more suitable since implementation languages are often state-based. Therefore, a formal method (or a set of collated methods) should cover both, event-based and state-based views and should provide a smooth transition between these views.

Petri nets are a formal model which is equally well suited for both views; *places* correspond to the state-based view and *transitions* correspond to the event-based view. Maybe, one of the main features of Petri nets is that they are perfectly balanced with respect to both views. Therefore, formal methods based on Petri nets allow a smooth transition between the event-based view and the state-based view. Most temporal logics introduced for Petri nets do not support both views — these logics are either state-based or event-based. In this paper we introduce a temporal logic which supports both views. This logic is an extension of a set of related state-based temporal logics for Petri nets

*supported by the DFG-Forschergruppe "Petri Net Technology"

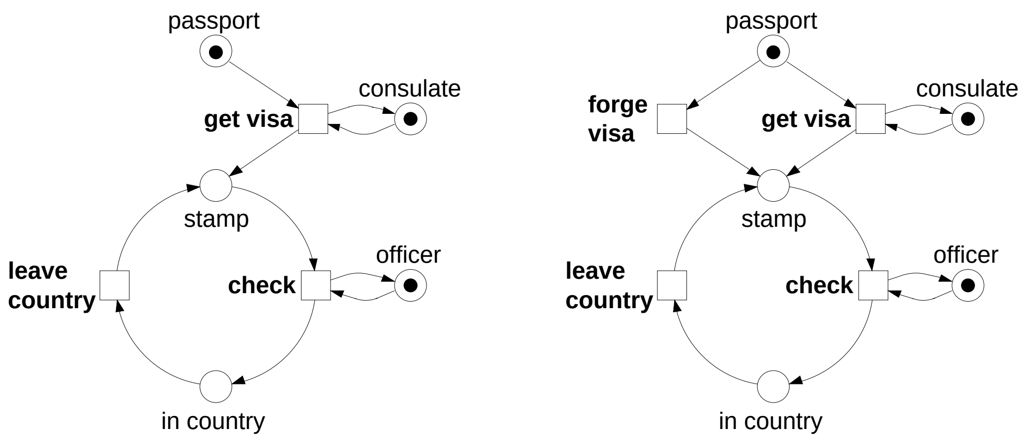
†e-mail: {kindler,vesper}@informatik.hu-berlin.de

[Rei95, KW95, WVV+96, Rei97b], which have been developed for the verification of distributed algorithms and are called DAWN[1] for this reason. The proposed extension of the logic can be used in the early design phases for formalizing informal requirements as well as in the later phases to verify state-based properties. Actually, this extension was inspired by informal requirements of the case study HDMS [CHL95], where many informal requirements are event-based.
The emphasis of this paper is on the introduction of the temporal operators and their interpretation on Petri nets. One feature of this interpretation is that state-based formulas from the original temporal logic preserve their original meaning; this way the original proof rules are still valid. The new rules presented in this paper should only provide a flavour of event-based rules; they are by no means complete. A set of adequate proof rules for this new logic is still under development.

The paper is organized as follows. First, we demonstrate by help of an example that sometimes an event-based notation allows a more faithful formalization of informal requirements. The basic definitions of nets and their runs will be introduced in Sect. 3. In Sect. 4 we introduce the temporal logic for states and events in a uniform setting. In Sect. 5 we present some simple verification techniques which are applied to the example.

## 2 A simple example

In order to demonstrate the benefits of an event-based logic, we present a simple example. The example shown in Fig. 1(a) models the usual procedure to enter a foreign country. First you take your passport and receive a visa at the consulate of the foreign country. Only with this visa (e.g. stamped into your passport) you are allowed to enter the country. After some time you leave the country. Note that we neglect the fact that usually a visa is valid only for a limited period.




(a) $\Sigma_1$ — An entry protocol (b) $\Sigma_2$ — An illegal entry protocol

Figure 1: Two examples

Now, the required property of this procedure is that you may only enter the country if you have been at the consulate before and received your visa. Often, this property is implicitly formalized[2] in a state-based way by

$$\Sigma_1 \models \Box\,(\mathsf{in\ country} \rightarrow \diamondminus\mathsf{stamp}) \tag{1}$$

Precisely, this formula means: Whenever (i.e. 'always' when) someone is in the country he must ('once') have had a stamp in his passport. This implicit formalization has a slight flaw. For example, consider

[1]Distributed Algorithms' Working Notation

[2]The temporal operators $\Box$ and $\diamondminus$ read 'always' and 'once'; they will be formally introduced in Sect. 4.

the net model shown in Fig. 1(b); this system satisfies the state-based formal requirement, but does not satisfy the informal requirement — no country wants a forger to enter the country. So, in contrast to the informal requirement the formalization "permits" forgery because the temporal formula is valid for the net with the additional transition $\mathsf{forge\ visa}$, too.
Therefore, we would like to have a notation which allows to explicitly express that someone who passes the control has got a visa before. To this end, we use an event-based temporal logic:

$$\Sigma_1 \models \boxed{\Box}\,(\mathsf{check} \ \rightarrow \Diamond\!\!\!\!\diamond\,\mathsf{get\ visa}\,) \tag{2}$$

In contrast to the state-based logic, we also allow transitions in the expressions; moreover, we use different but related temporal operators.
In this paper we formally define the meaning of these formulas and show how the state-based logics of [Rei95, KW95] smoothly extend to an event-based logic. Note that we use different symbols for the 'same' temporal operators in the event-based version and the state-based version because we want to integrate both views in a single temporal logic. The use of different operators allows for a nesting of event-based and state-based temporal formulas. Moreover, purely state-based formulas have the same meaning as before.

Before formalizing this logic, let us consider the above example again. Why does the state-based formalization not capture the informal representation? The reason is that in the informal specification the activities are taken for granted and transitions inscribed by these activities (e.g. by $\mathsf{get\ visa}$) are considered to be faithful. In contrast the inscriptions of a place (e.g. $\mathsf{stamp}$) is not faithful because it could be forged. If we would allow to inscribe the forge transition by $\mathsf{get\ visa}$ the event-based specification turns out to be as useless as the state-based formalization! This shows that there is no absolute argument in favour of event-based approaches; actually below each event-based view you can find a state-based view again (on a lower level of abstraction) and vice versa. This shows that there is no lowest level of abstraction which is event-based or state-based. Fortunately, we are not looking for the lowest level of abstraction but for the most adequate level of abstraction. We argue that often an event-based view is adequate to start with.

# 3 Basic Definitions

In this section we introduce the prerequisites for a formal definition of the logic. We define *Place/Transition-systems* (*P/T-systems*) — the version of Petri nets used in this paper — and their non-sequential *runs*. The definitions mainly follow the lines of [Rei85, BF88]. Section 3.1 introduces P/T-systems along with some standard Petri net notations. Section 3.2 formalizes the runs of a Petri net and some related concepts.

## 3.1 Place/Transition-Systems

Readers familiar with Place/Transition-Systems can skim this section; the only restrictions imposed to the standard definition are that all arcs are (implicitly) inscribed by 1, the initial marking is finite, and all transitions have non-empty and finite presets and postsets.

**Definition 3.1** Let $P$ and $T$ be two disjoint sets and $F$ be a relation, such that $F \subseteq (P \times T) \cup (T \times P)$. A triple $N = (P, T, F)$ is a *net*. The elements of $P$, $T$ and $F$ are called *places, transitions* and *arcs*, respectively.
A net $N$ is *T-restricted* iff for each transition $t \in T$ there exist places $p, q \in P$, such that $(p, t) \in F$ and $(t, q) \in F$ hold. A net is *finitely-branching*, iff for each transition $t \in T$ the set $\{p \in P : (p, t) \in F \text{ or } (t, p) \in F\}$ is finite.
A function $M : P \to \mathbb{N}$ is a *marking* of $N$. A pair $(N, M_0)$ is a *P/T-system* iff $N$ is a T-restricted and finitely-branching net and $M_0$ is a finite marking of $N$.

As usual, places, transitions and arcs are graphically represented by circles, squares, and arrows, respectively. A marking is graphically represented by the corresponding number of black dots (so called *token*) in places.

**Notation 3.2** Let $N = (P, T, F)$ be a net. For short, we write $x \in N$ for $x \in P \cup T$. For $x \in N$ the *preset* ${}^\bullet x$ and the *postset* $x^\bullet$ are defined by ${}^\bullet x = \{y \in N : (y, x) \in F\}$ and $x^\bullet = \{y \in N : (x, y) \in F\}$. The *minimal elements* ${}^\circ N$ and the *maximal elements* $N^\circ$ of a net $N$ are defined by ${}^\circ N = \{x \in N : {}^\bullet x = \varnothing\}$ and $N^\circ = \{x \in N : x^\bullet = \varnothing\}$.
A marking $M_1$ is *included in* a marking $M_2$, denoted by $M_1 \leqslant M_2$, iff for each $p \in P$ holds $M_1(p) \leq M_2(p)$. Let $M_1$ and $M_2$ be markings of a net $N$. We define the *addition* $M_1 + M_2$ and, if $M_2 \leqslant M_1$, the *subtraction* $M_1 - M_2$ of markings elementwise.

Where clear from the context, a set of places $Q \subseteq P$ also stands for the marking represented by the characteristic function of $Q$. This convention is used in the following definition for the preset ${}^\bullet t$ and the postset $t^\bullet$ of a transition $t$ for defining the firing rule.

**Definition 3.3** Let $N = (P, T, F)$ be a net and $M$ be a marking of $N$. A transition $t \in T$ is *enabled at* $M$ iff ${}^\bullet t \leqslant M$. A transition $t \in T$ which is enabled at $M$ may *occur*, leading to the *follower marking* $M'$, which is defined by $M' = (M - {}^\bullet t) + t^\bullet$. A marking $M'$ is *reachable from* a marking $M$ iff there exists a finite (possibly empty) sequence of occurrences of transitions leading from $M$ to $M'$. A marking $M$ is a *reachable marking of a P/T-system* $(N, M_0)$ iff it is reachable from $M_0$.

## 3.2 Runs of P/T-systems

Up to now, we have defined the system model. Next, we will define the behaviour of a system in terms of its runs. We use non-sequential runs of P/T-systems as defined by Golz and Reisig [GR83] for two reasons: First, non-sequential runs faithfully model concurrency. Second, in these runs the occurrence of both, places and transitions are represented in a natural way. Therefore we can easily define a logic for events and states based on these runs.
We start with an example: Figure 2 shows a run of $\Sigma_1$. Informally, a run is an unwinding of the P/T-system, where conflicts are resolved. Technically, a run is an *occurrence net* (i.e. an acyclic and conflict-free net) equipped with a labelling which establishes the correspondence between the P/T-system and the occurrence net. For a detailed motivation of the technical definitions of a run we refer to [GR83, BF88, KW95].

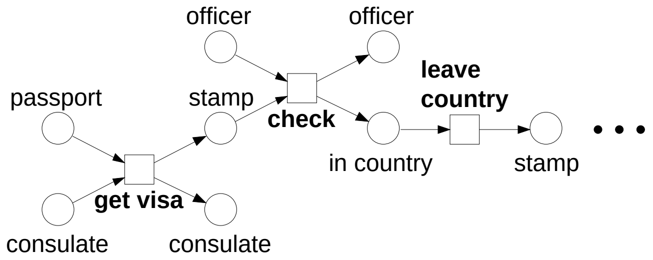



Figure 2: $R_1$ — A run of $\Sigma_1$

**Definition 3.4** A net $K = (B, E, \lessdot)$ is an *occurrence net* iff

1. for each $b \in B$ holds $|{}^\bullet b| \leq 1$ and $|b^\bullet| \leq 1$,
2. the transitive closure of $\lessdot$ (denoted by $<$ in the following) is acyclic, and
3. for each $x \in K$ the set of all predecessors of $x$ (i.e. $\{y \in K : y < x\}$) is finite.

A run of a P/T-system is an occurrence net together with a labelling function:

**Definition 3.5** Let $\Sigma = (N, M_0)$ be a P/T-system with $N = (P, T, F)$. Moreover, let $K = (B, E, \lessdot)$ be an occurrence net and $\varrho : B \cup E \to P \cup T$ be a labelling function. For a set $X \subseteq B \cup E$ we denote the set $\{\varrho(x) \,:\, x \in X\}$ by $\varrho(X)$. The pair $(K, \varrho)$ is a *run* of $\Sigma$ iff

1. $\varrho(B) \subseteq P$ and $\varrho(E) \subseteq T$,
2. for each $p \in P$ holds $M_0(p) = |\{b \in {}^{\circ}K \,:\, \varrho(b) = p\}|$,
3. for each $e \in E$ the restrictions of $\varrho$ to ${}^{\bullet}e$ and to $e^{\bullet}$ are injective; moreover we require ${}^{\bullet}\varrho(e) = \varrho({}^{\bullet}e)$ and $\varrho(e)^{\bullet} = \varrho(e^{\bullet})$, and
4. for each $t \in T$ the condition ${}^{\bullet}t \subseteq \varrho(K^{\circ})$ does not hold.

**Definition 3.6** Let $K = (B, E, \lessdot)$ be an occurrence net. A set $Q \subseteq K$ is a *co-set* of $K$ iff for each $q, r \in Q$ neither $q < r$ nor $r < q$ holds true. A co-set $Q$ is a *cut* of $K$ iff $Q$ is maximal, i.e. there exists no co-set $Q' \neq Q$ with $Q \subseteq Q'$. A cut $Q$ is a *state-cut* iff $Q \subseteq B$.
Let $C$ and $C'$ be finite cuts of an occurrence net $K$. $C'$ is *reachable from* $C$, denoted $C \preceq C'$, iff for each $c \in C$ there exists a $c' \in C'$, such that $c < c'$ or $c = c'$ holds.

Note that the reachability $C \preceq C'$ of finite state-cuts in a run of a system corresponds to the reachability of the corresponding markings in the P/T-system. This can be easily seen; the proof, however, requires some technical effort (cf. [BF88]).

# 4 Properties of P/T-systems

Now, we will formalize the temporal operators which are already known from the introduction. The operators $\Box$, $\Diamond$, $\boxminus$, and $\diamondminus$ are used in the usual meaning [MP92] and are read *always*, *eventually*, *so far*, and *once*, respectively. But, we interpret them on (non-sequential) runs as suggested in [Rei95, KW95]. These operators refer to state-cuts only. In order to formalize event-based properties, we introduce a corresponding set of temporal operators $\boxbox$, $\diamonddiamond$, $\boxboxminus$, and $\diamonddiamondminus$, which are interpreted on arbitrary cuts. These operators read *every-time*, *sometime*, *every-time in the past*, and *sometime in the past*, respectively. One benefit of introducing two versions of temporal operators is that purely state-based formulas keep their original meaning.

## 4.1 Syntax

In the formal definition of the syntax, we only introduce the operators $\Diamond$, $\diamondminus$, $\diamonddiamond$, and $\diamonddiamondminus$ because the other operators can be defined as dual versions. The atoms from which system properties can be built are the places and transitions of the P/T-system. A place in a system property indicates that the corresponding place occurs in the considered cut; a transition indicates that the transition occurs in the cut.

**Definition 4.1** Let $\Sigma = ((P, T, F), M_0)$ be a P/T-system. The set of *system properties* $SP$ is inductively defined by:

1. for each $p \in P$ holds $p \in SP$,
2. for each $t \in T$ holds $t \in SP$,
3. if $\varphi \in SP$ then $\neg\varphi \in SP$,
4. if $\varphi, \psi \in SP$ then $(\varphi \vee \psi) \in SP$,
5. if $\varphi \in SP$ then $\Diamond\varphi \in SP$,
6. if $\varphi \in SP$ then $\diamondminus\varphi \in SP$,
7. if $\varphi \in SP$ then $\diamonddiamond\varphi \in SP$,
8. if $\varphi \in SP$ then $\diamonddiamondminus\varphi \in SP$.

**Notation 4.2** The operators $\wedge$ and $\to$ are the usual abbreviations. The operators $\Box\varphi$, $\boxbox\varphi$, $\boxminus\varphi$, and $\boxboxminus\varphi$ are abbreviations (dual versions) for $\neg\Diamond\neg\varphi$, $\neg\diamonddiamond\neg\varphi$, $\neg\diamondminus\neg\varphi$, and $\neg\diamonddiamondminus\neg\varphi$, respectively.

### 4.2 Semantics

Now, we define the semantics of a *system property*. To this end, we define when a *system property* holds in a finite cut of a run. A P/T-system satisfies a system property, if the property holds in the *initial cut* of each run of the system.

**Definition 4.3** Let $\Sigma = ((P, T, F), M_0)$ be a P/T-system, $R = (K, \varrho)$ be a run of $\Sigma$, and $C$ be a finite cut of $R$. We define the *validity of a system property* inductively as follows:

1. $C \models p$ iff $p \in \varrho(C)$,
2. $C \models t$ iff $t \in \varrho(C)$,
3. $C \models \neg\varphi$ iff $C \models \varphi$ does not hold,
4. $C \models (\varphi \vee \psi)$ iff $C \models \varphi$ or $C \models \psi$ holds,
5. $C \models \Diamond\varphi$ iff there exists a finite state-cut $C'$, such that $C \preceq C'$ and $C' \models \varphi$ holds,
6. $C \models \diamondminus\varphi$ iff there exists a finite state-cut $C'$, such that $C' \preceq C$ and $C' \models \varphi$ holds,
7. $C \models \doublediamond\varphi$ iff there exists a finite cut $C'$, such that $C \preceq C'$ and $C' \models \varphi$ holds,
8. $C \models \doublediamondminus\varphi$ iff there exists a finite cut $C'$, such that $C' \preceq C$ and $C' \models \varphi$ holds.

A system property $\varphi \in SP$ is *valid in run* $R$ (denoted by $R \models \varphi$) iff ${}^{\circ}K \models \varphi$ holds, it is *valid in* $\Sigma$ (denoted by $\Sigma \models \varphi$) iff $\varphi$ is valid in all runs of $\Sigma$.

Let us consider some examples of system properties:

$$\Box\,(\mathsf{passport} \vee \mathsf{stamp} \vee \mathsf{in\ country})$$

$$\boxbox\,(\mathsf{passport} \vee \mathsf{stamp} \vee \mathsf{in\ country})$$

$$\boxbox\,(\mathsf{passport} \vee \mathsf{get\ visa} \vee \mathsf{stamp} \vee \mathsf{check} \vee \mathsf{in\ country} \vee \mathsf{leave\ country})$$

The first one and the third one are valid in $R_1$ and $\Sigma_1$ (see Fig. 2 and 1(a)), the second one is not valid in $R_1$ and $\Sigma_1$ because in $R_1$ there exists a non-state-cut containing get visa and officer, only. So, the first and the second example show the essential difference between the operators $\Box$ and $\boxbox$.
In the state-based logic the first property can be easily proven by help of a *place invariant* (e.g. [KW95]). This technique also works for the purely state-based formulas of the extended logic. The third property shows a generalization of the concept of place invariants; we need to add transitions to the formula which may occur 'between' two states.
The above definitions immediately imply the following rules:

**Lemma 4.4** Let $\Sigma$ be a P/T-system and $\varphi \in SP$ a system property. Then we have:

1. If $\Sigma \models \Diamond\varphi$ then $\Sigma \models \doublediamond\varphi$,
2. If $\Sigma \models \diamondminus\varphi$ then $\Sigma \models \doublediamondminus\varphi$,
3. If $\Sigma \models \boxbox\,\varphi$ then $\Sigma \models \Box\,\varphi$,
4. If $\Sigma \models \boxed{\boxminus}\,\varphi$ then $\Sigma \models \boxminus\,\varphi$.

In this paper the atoms of the temporal logic are quite simple; the only legal atoms are places and transitions. For more sophisticated applications more complex atoms are necessary. For example, we would like to write $p_1 + p_2 + p_3 = 1$ (cf. [KW95]) for places $p_1$, $p_2$, and $p_3$. It is straightforward to extend the language of legal atoms. Here, we concentrate on the temporal operators.

# 5 Proof Rules

In this section we will provide some proof rules for system properties. Since the state-based fragment of the temporal logic is semantically equivalent to the purely state-based logic (cf. [MP92, Rei95, WVV$^+$96]) the corresponding rules are still valid. Therefore, we concentrate on rules for event-based properties and rules which relate both views. Still, we can only give some selected rules, which are sufficient for our example.
The first proof rule is based on a simple observation: If a transition $t$ occurs then its preset was marked before.

**Proposition 5.1** Let $\Sigma = ((P, T, F), M_0)$ be a P/T-system and $t \in T$ be a transition of $\Sigma$. Then we have

$$\Sigma \models \boxbox \left(t \rightarrow \Diamond \bigwedge_{p \in {}^\bullet t} p\right)$$

We can apply this rule to our introductory example (see Fig. 1(a)), with $t = \mathsf{check}$ :

$$\Sigma_1 \models \boxbox \left(\mathsf{check} \rightarrow \Diamond\, (\mathsf{officer} \wedge \mathsf{stamp})\right) \tag{3}$$

The second rule is based on the following observation: If we know that a currently marked place has been initially unmarked then a transition in its preset must have occurred before.

**Proposition 5.2** Let $\Sigma = ((P, T, F), M_0)$ be a P/T-system and $p \in P$ such that $M_0(p) = 0$. Then we have

$$\Sigma \models \boxbox \left(p \rightarrow \diamonddiamond \bigvee_{t \in {}^\bullet p} t\right)$$

Now we will apply this rule to our example. We know that initially the place $\mathsf{stamp}$ is not marked. The application of rule 5.2 gives us:

$$\Sigma_1 \models \boxbox \left(\mathsf{stamp} \rightarrow \diamonddiamond (\mathsf{get\ visa} \vee \mathsf{leave\ country})\right) \tag{4}$$

From (3) and the standard weakening rule follows:

$$\Sigma_1 \models \boxbox \left(\mathsf{check} \rightarrow \Diamond \mathsf{stamp}\right) \tag{5}$$

In combination with (4), Lemma 4.4.2, and a $\diamonddiamond$-transitivity rule (e.g. [MP92], p. 265) follows:

$$\Sigma_1 \models \boxbox \left(\mathsf{check} \rightarrow \diamonddiamond\, (\mathsf{leave\ country} \vee \mathsf{get\ visa})\right) \tag{6}$$

An analogous application of rule 5.1 and rule 5.2 leads to the following property of $\Sigma_1$:

$$\Sigma_1 \models \boxbox \left(\mathsf{leave\ country} \rightarrow \diamonddiamond \mathsf{check}\right) \tag{7}$$

Let us consider properties (6) and (7) together. We know from Def. 3.4.3 that each occurrence of transition $\mathsf{check}$ has a finite history. Property (6) ensures that in the past there exists an event labelled by $\mathsf{leave\ country}$ or an event labelled by $\mathsf{get\ visa}$. In the first case there exists another event labelled by $\mathsf{check}$ in its past (7). Because of the finite history there also exists an event labelled by $\mathsf{get\ visa}$ in its past. The following proof rule formalizes this observation. Note that this proof rule holds in a purely state-based proof system as well.

**Proposition 5.3** Let $\Sigma = ((P, T, F), M_0)$ be a P/T-system. Let $\varphi, \psi, \chi \in SP$ be system properties of $\Sigma$. If the conditions $\Sigma \models \boxbox \left(\varphi \rightarrow \diamonddiamond\, (\psi \vee \chi)\right)$, $\Sigma \models \boxbox \left(\chi \rightarrow \diamonddiamond \varphi\right)$, and $\Sigma \models \boxbox \neg (\varphi \wedge \chi)$ hold, then the following holds, too:

$$\Sigma \models \boxbox \left(\varphi \rightarrow \diamonddiamond \psi\right)$$

We can apply this proof rule to $\Sigma_1$: Let $\varphi = \mathsf{check}$, $\chi = \mathsf{leave\ country}$, and $\psi = \mathsf{get\ visa}$. Properties (6) and (7) ensure the validity of the first two assumptions of rule 5.3. It remains to show

$$\Sigma_1 \models \Box \neg (\mathsf{check} \wedge \mathsf{leave\ country}) \tag{8}$$

which can be proven by a generalization of place invariants, where also transitions are taken into account[3]. Now we can apply rule 5.3 which finishes the proof:

$$\Sigma_1 \models \Box \big(\mathsf{check} \rightarrow \Diamond \mathsf{get\ visa}\big) \tag{9}$$

Though these these rules are very restricted, they are sufficient to prove a slightly extended example. Consider the P/T-system $\Sigma_3$ from Fig. 3. Structurally, it would be possible for a criminal to forge a

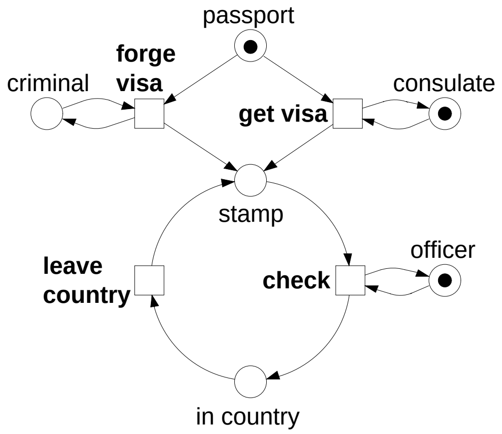



Figure 3: $\Sigma_3$ — Another entry protocol

visa. Fortunately, there is no criminal. Therefore, the runs of $\Sigma_1$ and $\Sigma_3$ are the same. This means that the same system properties hold for $\Sigma_1$ and $\Sigma_3$, which is in particular true for property (2). But how about a proof? Following analogous proof steps we obtain:

$$\Sigma_3 \models \Box \big(\mathsf{check} \rightarrow \Diamond (\mathsf{get\ visa} \vee \mathsf{forge\ visa})\big) \tag{10}$$

Furthermore, we can pick up the following property easily[4]:

$$\Sigma_3 \models \Box \neg \mathsf{forge\ visa} \tag{11}$$

By the standard weakening rule we get:

$$\Sigma_3 \models \Box \big(\mathsf{forge\ visa} \rightarrow \Diamond \mathsf{check}\big) \tag{12}$$

Together with (10) we can apply rule 5.3 again and we obtain (2).

The proof rules 5.1 and 5.2 allow verification of typical safety properties which occur in many case studies. Analogous proof rules for *leads-to* properties exist as well. Leads-to properties are particular liveness properties which are well suited for the specification of distributed algorithms ([Rei97b, Rei97a, WVV$^+$96]). In terms of our temporal logic, leads-to properties are system properties of the form: $\Box \big(\varphi \rightarrow \Diamond \psi\big)$. Here we show that a simple form of the *progress rule* from [KW95] can be decomposed into two even simpler rules: The first rule says that an enabled transition will either eventually occur or a transition in conflict will occur. The second rule says that when a transition occurs its postset will eventually be marked.

[3]The generalization would read $\Box$ ($\mathsf{passport} + \mathsf{get\ visa} + \mathsf{stamp} + \mathsf{check} + \mathsf{in\ country} + \mathsf{leave\ country} = 1$); but as we did not introduce these atoms, we do not formalize this rule either.

[4]Again, this follows from the generalized place invariant $\Box$ ($\mathsf{criminal} + \mathsf{forge\ visa} = 0$).

**Proposition 5.4** Let $\Sigma = ((P,T,F),M_0)$ be a P/T-system and $t \in T$. Then, the following holds:

$$\Sigma \models \boxdot\Big(\bigwedge_{p\in{}^\bullet t} p \to \Diamond\!\!\!\Diamond \bigvee_{t'\in({}^\bullet t)^\bullet} t'\Big) \qquad \Sigma \models \boxdot\Big(t \to \Diamond \bigwedge_{p\in t^\bullet} p\Big)$$

The two parts of this proposition resemble the *progress property* which guarantees that an activated transition either eventually occurs itself or it eventually becomes disabled by the occurrence of a conflicting transition.

## 6 Conclusion

In this paper we have extended a state-based temporal logic for Petri nets by events. This logic enjoys almost the same balance between events and states as Petri nets themselves. Therefore, the logic is equally well suited for formalizing requirements from the state-based view and from the event-based view, and provides a smooth transition between them. This feature allows a smooth transition between different system development phases.

Here we presented a logic for P/T-systems and gave a rough sketch of some proof rules. For realistic design of distributed system which incorporate data we need a logic for high-level Petri nets — in particular in the late phases of the development process. The logic presented in this paper can be easily adapted to algebraic Petri nets. In order to deal with transitions in different modes in a suitable way, the transitions must be inscribed by an expression (e.g. check.person, where person is a variable, which receives the value of the corresponding occurrence mode of the transitions). This way, we can tell which person enters a country. A formal presentation of this extension of algebraic Petri nets can be found in [KW96]. Then, the condition when a person x may enter the country would read $\boxdot$ check.x $\to \Diamond\!\!\!\Diamond$ get visa.x . More details will be presented in a forthcoming paper.

The starting point of this work have been examples of informal requirements of a system under development. In some cases the event-based logic allowed a more faithful formalization of these requirements. When we extended the logic we realized that in this extended logic some properties which are very hard to prove in the state-based logic (e.g. the example from [PP94] which is hard to prove [Rei97b]) can now easily be proven. In the event-based extension of the temporal logic some purely state-based properties have much nicer and more natural proofs; this is also suggested by the decomposition of the progress rule of the state-based logic into two simpler parts in the event-based extension. Though not the goal of the event-based extension, we believe that it has better chances for a positive completeness result — which is subject to further research.

## References


[BF88] Eike Best and César Fernández. *Nonsequential Processes*, *EATCS Monographs on Theoretical Computer Science* 13. Springer-Verlag, 1988.

[CHL95] F. Cornelius, H. Hußmann, and M. Löwe. The KORSO case study for software engineering with formal methods: A medical information system. In M. Broy and S. Jähnichen, editors, *KORSO: Methods, Languages, and Tools for the Construction of Correct Software*, *LNCS* 1009, pages 417–445. Springer-Verlag, 1995.

[EGVW97] C. Ermel, M. Gajewsky, T. Vesper, and M. Weber. Verifikation strukturierter Netze. In H. Weber, H. Ehrig, and W. Reisig, editors, *Proceedings des Move-On-Workshop der DFG-Forschergruppe Petrinetz-Technologie*. TU-Berlin, August 1997. to appear.

[GR83] Ursula Goltz and Wolfgang Reisig. The non-sequential behaviour of Petri nets. *Information and Control*, 57:125–147, 1983.

[KW95] 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.

[KW96] Ekkart Kindler and Rolf Walter. Arc-typed Petri nets. In J. Billington and W. Reisig, editors, *Application and Theory of Petri Nets 1996*, *LNCS* 1091, pages 289–306. Springer-Verlag, June 1996.

[MP92] Zohar Manna and Amir Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification.* Springer-Verlag, 1992.

[PP94] Doron Peled and Amir Pnueli. Proving partial order properties. *Theoretical Computer Science*, 126:143–182, 1994.

[Rei85] Wolfgang Reisig. *Petri Nets*, *EATCS Monographs on Theoretical Computer Science* 4. Springer-Verlag, 1985.

[Rei95] Wolfgang Reisig. Petri net models of distributed algorithms. In J. van Leeuven, editor, *Computer Science Today: Recent Trends and Developments*, *LNCS* 1000, pages 441–454. Springer-Verlag, Berlin, 1995.

[Rei97a] Wolfgang Reisig. *Elements of Distributed Algorithms: Modelling and Analysis with Petri Nets.* In preparation, 1997.

[Rei97b] Wolfgang Reisig. Interleaved progress, concurrent progress, and local progress. In D.A. Peled, V.R. Pratt, and G.J. Holzmann, editors, *Partial Order Methods in Verification*, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 29, pages 99–115. AMS, 1997.

[vdA96] W.M.P. van der Aalst. Three good reasons for using a Petri-net-based workflow management system. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises*, pages 179–201, Cambridge, Massachusetts, November 1996.

[WVV+96] R. Walter, H. Völzer, T. Vesper, W. Reisig, E. Kindler, J. Freiheit, and J. Desel. Memorandum: Petrinetzmodelle zur Verifikation Verteilter Algorithmen. Informatik-Bericht 67, Humboldt-Universität zu Berlin, July 1996.