Computing and Informatics, Vol. 37, 2018, 1464–1484, doi: 10.4149/cai_2018_6_1464

ACP SEMANTICS FOR PETRI NETS

Slavomír Šimoňák, Martin Tomášek

Department of Computers and Informatics
Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
e-mail: {slavomir.simonak, martin.tomasek}@tuke.sk

Abstract. The paper deals with algebraic semantics for Petri nets, based on process algebra ACP. The semantics is defined by assigning a special variable to every place of given Petri net, expressing the process initiated in the place. Algebraic semantics of the Petri net is then defined as a parallel composition of all the variables, where corresponding places hold tokens within the initial marking. Resulting algebraic specification preserves operational behavior of the original net-based specification.

Keywords: Petri nets, process algebra ACP, formal methods, semantics, specification, transformation

Mathematics Subject Classification 2010: 68-Q85

1 INTRODUCTION

In formal methods community a widely accepted assertion states that a single formal method, that will cover all aspects of the system design process in an acceptable way, will never be developed [26]. That is given by the complexity of the process and a variety of systems to be designed and analyzed. There have been attempts to integrate two or more formal methods to cope with that situation. According to [7] it is particularly fruitful to study combinations of several methods with different characteristics and complementary strengths. A similar approach has been applied at the authors' home institution regarding the development of an environment (termed mFDTE) [21] for the design and analysis of discrete systems based on integration of three formal description techniques: Petri nets, process algebra and B-method.

The choice of the techniques mentioned has been driven by their properties that in a complementary way cover different aspects of the system design and analysis. In this work we pay attention to two of them: Petri nets and process algebra.

Petri nets, as a widely accepted formalism for design and analysis of discrete systems [24], have proved their qualities in many situations [15, 33]. Both the states and the actions of the system are clearly described using a Petri net model. So there are analysis techniques [10] available for investigation of properties based on states as well as on the dynamic behavior of a model. Process algebraic specification of a system usually has no explicit representation of states, and is more focused on the description of its dynamic behavior. Techniques available in process algebra are especially useful when comparing the behavioral descriptions of concurrent systems, like a high-level system specification and a more detailed description of its implementation [7]. In addition to the differences mentioned above, the de/composition of specifications in the case of Petri nets is not so natural and fluent as it is in the case of process algebra. Consequently, it can be assumed that Petri nets and process algebra meet the property of complementarity in several aspects.

In the recent years an active research has been performed in the matter of two fundamental models of concurrency – Petri nets and process algebras [5]. In [17] a CCS-like calculus named as Finite-Net Multi-CCS was introduced. The given calculus is provided by a labeled transition system and the P/T net semantics. As a consequence, well-formed Finite-Net Multi-CCS processes are able to represent finite P/T nets [18]. A simple process calculus (Petri calculus) is defined in [27]. Translation from Petri nets with boundaries to the calculus, and also translation of Petri calculus terms to Petri nets is introduced. In such a way, the same expressiveness of both formalisms is shown. Another interesting work [22] compares a range of notions for treatment of concurrency, such as Petri nets, Mazurkiewicz trace languages and Zielonka automata to process algebra and raises several interesting questions.

Further, a selection of classical but very influential works on the topic is given. In [25] relations between nets, terms and formulas are treated. Particularly, the net semantics of terms and the process semantics of nets are defined. In [11] relations between the process algebra PBC (Petri Box Calculus) and a class of P/T nets are studied. Syntax and semantics of PBC terms are carefully selected to allow the definition of a transformation yielding the P/T nets preserving structural operational semantics of the source terms. The transformation allows a composition of P/T nets. A process semantics of elementary nets is defined using concepts of partial algebra in [13]. In the paper, a partial algebra is proposed, as a suitable tool for defining the true-concurrency semantics for arbitrary restrictions of the occurrence rule. In the work [4] authors treat the issue of partial-order algebras and their relations to P/T nets based on the theory of BPA and ACP. Authors of [6] propose an approach to algebraic semantics for hierarchical P/T nets. The PTNA (Place/Transition Net Algebra) is defined within the paper, based on process algebra ACP and an algebraic semantics for P/T nets is given, such that a P/T net and the term representation of the net have the same operational behavior. The

actions of the process algebra correspond to consumption and production of tokens by transitions respectively. The results achieved are further extended to hierarchical P/T nets.

The compositional approach to system design and the possibility of investigating and comparing the system properties by means of algebraic manipulations [27, 17] are two main sources of motivation for defining the algebraic semantics for Petri nets. This work aims to contribute to the topic of integrating Petri nets and process algebra by introducing the algebraic semantics for Petri nets, based on Algebra of Communicating Process (ACP) [2]. Our previous work on integration of Petri nets and process algebra includes the definition of algebraic semantics for Petri nets based on a dedicated process algebra APC (Algebra of Process Components) [30]. While it has several interesting properties, mainly the ability to model Petri nets processes in a simple and natural way, it also has one essential drawback, which hampers its practical utilization. There is currently no available tool supporting the analysis of APC specifications. This motivated us to define the algebraic semantics for Petri nets based on process algebra with a reasonable tool support. The available tool support in this case includes the PSF-Toolkit [14], which is based on the process algebra ACP.

When comparing the proposed solution with the presented existing solutions, several differences can be identified. We use a widely-adopted process algebra without defining its special extensions, we can find them in many existing solutions, which ensures the reasonable tool support out of available tools. Moreover, the existing implementation of our transformation supports the simpler practical utilization. Most of the solutions, except of those defining their own process language, are based on process algebras like CCS and CSP. We have chosen the process algebra ACP, as we believe it is beneficial. ACP is based on an equational style of reasoning, while CCS and CSP are model-based. In the case of ACP, the central point is an equational theory, which may have several semantic interpretations [7, 3].

The paper is organized as follows. Section 1 introduces the topic and the motivation for the work. Basic notions and definitions for the class of Petri nets used are given in Section 2. In Section 3, Algebra of Communicating Process is presented briefly from the syntactic as well as the semantic point of view. Section 4 concentrates on defining the algebraic semantics for given class of Petri nets. The example presented in Section 5 demonstrates the approach introduced within the paper. Section 6 concludes the paper and contains the possible directions for future research.

2 PETRI NETS

In the paper we assume the class of ordinary Petri nets [9] and provide a brief description of the basic notions and notations in the following paragraphs in style of [20].

Definition 1. The Petri net is a 4-tuple $N = (P, T, pre, post)$, where P is a finite set of places, T is a finite set of transitions ($P \cap T = \emptyset$), $pre : P \times T \to \{0, 1\}$ is the preset function and $post : P \times T \to \{0, 1\}$ is the postset function.

The marking of Petri net $N = (P, T, pre, post)$ is a totally defined function $m : P \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. We use m to describe a configuration of N. We fix some ordering of places ($P = \{p_1, \ldots, p_k\}$), so we can consider m to be a k-dimensional nonnegative integer vector: $\vec{m} \in \mathbb{N}^k$. More formally $\vec{m} = (m(p_1), m(p_2), \ldots, m(p_k))$, where $m(p_i)$ is the value of m in the place p_i, $i = 1, 2, \ldots, k$. Marked net, with the marking m, is denoted by $N_0 = (N, m_0)$ or $N_0 = (P, T, pre, post, m_0)$.

For the sake of simplicity we use the denotation m for both interpretations of the marking, when it does not cause any problems. Some useful notations can be further defined, as the sets of pre/post-conditions for given transition $t \in T$ and the sets of pre/post-transitions for given place $p \in P$, respectively:

$$
\begin{aligned}
&{}^\bullet t = \{p | pre(p, t) \neq 0\} \text{ the set of preconditions of } t,\\
&t^\bullet = \{p | post(p, t) \neq 0\} \text{ the set of postconditions of } t,\\
&p^\bullet = \{t | pre(p, t) \neq 0\},\\
&{}^\bullet p = \{t | post(p, t) \neq 0\}.
\end{aligned}
$$

We say that a transition t is enabled in m (and denote it $m \xrightarrow{t}$), if for every $p \in {}^\bullet t, m(p) \geq pre(p, t)$. The effect of firing t in m is the creation of the new marking m' ($m \xrightarrow{t} m'$) and m' is defined in the following way:

$$m'(p) = m(p) - pre(p, t) + post(p, t), p \in P, t \in T.$$

Denotation $(N, m) \xrightarrow{t} (N, m')$ is alternatively used for expressing a step of computation ($m \xrightarrow{t} m'$) within the Petri net N. The set of reachable markings for given Petri net $N_0 = (P, T, pre, post, m_0)$ is defined by:

$$\mathcal{R}(N_0) = \{m | m_0 \xrightarrow{\sigma} m\}$$

where $\sigma = t_1, t_2, \ldots, t_r$ stands for an admissible firing sequence in N_0. The language of Petri net N_0 can be defined by:

$$\mathcal{L}(N_0) = \{\sigma \in T^* | m_0 \xrightarrow{\sigma} m\}.$$

3 ACP – ALGEBRA OF COMMUNICATING PROCESSES

Algebra of Communicating Processes (ACP) [2] is an algebraic framework for studying concurrent communicating processes. It is based on Milner's Calculus of Communicating Systems (CCS) [23], historically the first complete theory. ACP more

emphasizes the algebraic aspect, it is an equational theory with a number of semantical models. In addition, the ACP uses a more general communication scheme compared to CCS and CSP [3].

3.1 Syntax

The signature of ACP contains a set of constants A, partial communication function γ on A, which is commutative and associative, a special constant δ (deadlock $\delta \notin A$) and operators: $+$ (alternative composition), $\cdot$ (sequential composition), $\|$ (parallel composition), $\lfloor\!\lfloor$ (left merge), $|$ (communication merge).

The axiom system of process algebra ACP can be found in Table 1. We also use axioms for encapsulation (D1–D4) and hiding internal transitions (TI1–TI4). Within the table x, y, z stand for processes, $a, b \in A$ and $\delta, \tau \notin A$ are constants. In [2], where many different process algebras are defined, it can be found with the denotation ACP$^\tau$, but for the sake of simplicity we use the name ACP within the paper.

$x + y = y + x$	A1	$\partial_H(a) = a$ if $a \notin H$	D1
$(x + y) + z = x + (y + z)$	A2	$\partial_H(a) = \delta$ if $a \in H$	D2
$x + x = x$	A3	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	D3
$(x + y) \cdot z = xz + yz$	A4	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$	D4
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	A5		
$x + \delta = x$	A6	$\tau_I(a) = a$ if $a \notin I$	TI1
$\delta \cdot x = \delta$	A7	$\tau_I(a) = \tau$ if $a \in I$	TI2
		$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	TI3
$a\|b = \gamma(a, b)$ if γ defined	CF1	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$	TI4
$a\|b = \delta$ otherwise	CF2		
		$x\tau = x$	B1
$x\|y = x\lfloor\!\lfloor y + y\lfloor\!\lfloor x + x\|y$	CM1	$x(\tau(y + z) + y) = x(y + z)$	B2
$a\lfloor\!\lfloor x = ax$	CM2		
$ax\lfloor\!\lfloor y = a(x\|y)$	CM3		
$(x + y)\lfloor\!\lfloor z = x\lfloor\!\lfloor z + y\lfloor\!\lfloor z$	CM4		
$ax\|b = (a\|b) \cdot x$	CM5		
$a\|bx = (a\|b) \cdot x$	CM6		
$ax\|by = (a\|b) \cdot (x\|y)$	CM7		
$(x + y)\|z = x\|z + y\|z$	CM8		
$x\|(y + z) = x\|y + x\|z$	CM9		

Table 1. Axioms of process algebra ACP

3.2 Semantics

The constants in A are called atomic actions, and are considered indivisible actions (events). The sequential composition of two processes $x \cdot y$ is the process that first

executes x and after finishing it, it starts executing y. The alternative composition of two processes x and y is the process $x + y$, that either executes x or y.

The parallel composition of two processes x and y in a process algebra without communication is expressed as $x \| y$. It is assumed that the atomic actions have no duration in time, and that two actions cannot happen simultaneously [2]. So the actions of x are arbitrarily interleaved with those of y. In order to specify the merge by finite number of equations, the auxiliary operator $\lfloor\!\lfloor$ is introduced. The operator has the same meaning as the merge operator ($\|$) with the restriction, that the first step must come from the left process (i.e. the x in case of $x \lfloor\!\lfloor y$).

In process algebra ACP however, the meaning of parallel composition is a bit more complicated. Considering the merge of two processes $x \| y$, there are three possibilities to proceed in general (CM1). Either the process starts with a first step of x (given by $x \lfloor\!\lfloor y$), or a first step of y ($y \lfloor\!\lfloor x$) or with a communication between the two processes ($x|y$).

The communication merge ($x|y$) represents the merge of two processes, where the first step is a communication between x and y. This type of merge is an extension of communication function on atomic actions ($\gamma : A \times A \to A$). When the communication function is not defined, the communication merge is equal to δ.

If we want to state that some actions cannot happen and should be blocked, the actions are renamed to δ by using the unary encapsulation operator ∂_H. A process $\partial_H(p)$ can execute all the actions of p, except those, which names are in the set H. An important tool in the analysis of systems is hiding, as the names of internal events can be hidden and thus the relationship between externally visible events becomes more clear. The hidden action (τ) cannot be observed directly, or communicate with other actions [16].

a)	$a \xrightarrow{a} \surd$	
b)	$x \xrightarrow{a} \surd \Rightarrow$	$x + y \xrightarrow{a} \surd$ and $y + x \xrightarrow{a} \surd$
		$x \cdot y \xrightarrow{a} y$
		$x \| y \xrightarrow{a} y$, $y \| x \xrightarrow{a} y$ and $x \lfloor\!\lfloor y \xrightarrow{a} y$
c)	$x \xrightarrow{a} x' \Rightarrow$	$x + y \xrightarrow{a} x'$ and $y + x \xrightarrow{a} x'$
		$x \cdot y \xrightarrow{a} x' \cdot y$
		$x \| y \xrightarrow{a} x' \| y$, $y \| x \xrightarrow{a} y \| x'$, $x \lfloor\!\lfloor y \xrightarrow{a} x' \| y$
d)	$x \xrightarrow{a} x'$, $y \xrightarrow{b} y'$, $\gamma(a,b) = c \Rightarrow$	$x \| y \xrightarrow{c} x' \| y'$ and $x \| y \xrightarrow{c} x' \| y'$
	$x \xrightarrow{a} x'$, $y \xrightarrow{b} \surd$, $\gamma(a,b) = c \Rightarrow$	$x \| y \xrightarrow{c} x'$, $x \| y \xrightarrow{c} x'$, $y \| x \xrightarrow{c} x'$, $y \| x \xrightarrow{c} x'$
	$x \xrightarrow{a} \surd$, $y \xrightarrow{b} \surd$, $\gamma(a,b) = c \Rightarrow$	$x \| y \xrightarrow{c} \surd$ and $x \| y \xrightarrow{c} \surd$

Table 2. Transition relations for ACP terms

To assign an operational semantics to process expressions, we determine, which actions the process can perform. The fact, that process represented by the term t can execute the action a and turn to the term s is denoted by: $t \xrightarrow{a} s$ (or alternatively a is enabled in t). The symbol $\surd$ stands for successful termination and thus $t \xrightarrow{a} \surd$

denotes the fact that t can terminate by executing a. The definition of action relations is given in Table 2.

4 ACP SEMANTICS FOR PETRI NETS

In this section the transformation is described in more detail. We start with defining a special variable for every place in a Petri net to be transformed. We name such variable as E-variable, and it will be bound to a term representing all possible computations started in corresponding place of the Petri net N. The value (term) assigned to the particular variable depends on the structure of the net in a vicinity of associated place. So, considering the place p, the variable $E(p)$ will be bound to a term representing all the computations within the Petri net N, which are initiated in the place p.

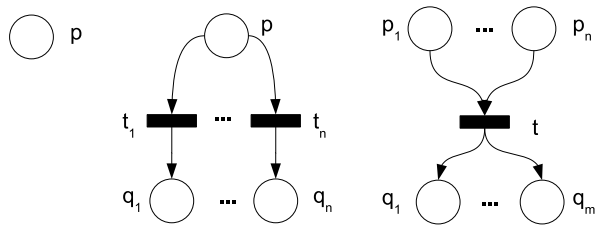


Figure 1. Petri net fragments

The basic configurations are captured in Figure 1. In the case a) a situation is depicted, where no arcs are connected to the place named p. This situation results in the assignment of a term representing no computations to the variable corresponding to such place, i.e. δ (deadlock). The case b) represents the alternative composition (choice). If a token is present in the place p, a choice is to be made, and only one of the transitions $t_1, \ldots, t_n$ can fire. The case c) represents a general composition, where tokens must be present in all the pre-places $(p_1, \ldots, p_n)$ of the transition t to enable it for firing. Otherwise the firing of the transition is not possible. After firing of a corresponding transition, however its post-place(s) are marked and the processes initiated in those places are able to start.

The general composition (the case c) of Figure 1) can be understood as a generalization of three basic kinds of composition: sequential, parallel and synchronization (see Figure 2). The fourth of the basic composition mechanisms being the alternative composition as it was mentioned above. Let n be the number of pre-places and m the number of post-places of the transition t, then:

- if $n = 1 \wedge m = 1$ we obtain the sequential composition (Figure 2 a)),
- if $n = 1 \wedge m > 1$ we obtain the parallel composition (Figure 2 b)),

- if $n > 1 \wedge m = 1$ we obtain the synchronization (Figure 2 c)).

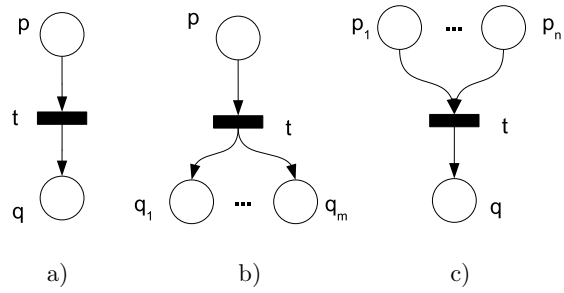


Figure 2. Basic compositions as a special cases of general composition

Now we can proceed to construction of terms representing the possible computations for particular types of places. When constructing the algebraic semantics for a Petri net, the terms will be bound to corresponding E-variables.

Definition 2. According to the structure of a Petri net in the vicinity of a given place, the bounding of terms to corresponding variables for elementary situations depicted in Figures 1, 2 and 3 is defined as follows:

1. deadlock (Figure 1 a)): $E(p) = \delta$,
2. alternative composition (Figure 1 b)): $E(p) = t_1 \cdot E(q_1) + t_2 \cdot E(q_2) + \ldots + t_n \cdot E(q_n)$,
3. sequential composition (Figure 2 a)): $E(p) = t \cdot E(q)$,
4. parallel composition (Figure 2 b)): $E(p) = t \cdot (E(q_1) \parallel \ldots \parallel E(q_m))$,
5. synchronization (Figure 2 c)): $E(p_1) = t_{p1}$, $E(p_2) = t_{p2}$, $\ldots$, $E(p_n) = t_{pn}$,
6. transition without post-place(s) (Figure 3 a)): $E(p) = t$,
7. transition without pre-place(s) (Figures 3 b) and 3 c)): $E(p) = t \cdot (E(p) \| E(q))$.

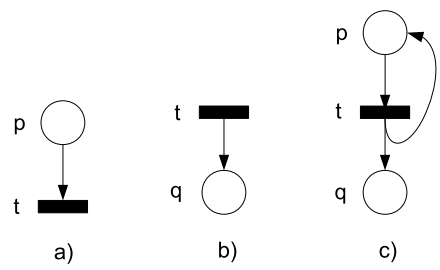


Figure 3. Transitions without input/output

We further give additional notes to the case 5 (synchronization) and the case 7 (transition without pre-place(s)) of the Definition 2. In the case of synchronization [1], the new auxiliary synchronization process is also defined in the following way:

$$Sync_t = (t_{s1} \| t_{s2} \| \dots \| t_{sn}) \cdot t \cdot (E(q) \| Sync_t), \tag{1}$$

ensuring that the synchronization must precede the execution of the action t. To achieve the goal, the following communication functions are also defined: $t_{s1}|t_{p1} = t_{c1}, t_{s2}|t_{p2} = t_{c2}, \dots, t_{sn}|t_{pn} = t_{cn}$. A set of actions to hide (rename to internal action τ), is defined to make the additional synchronization actions invisible: $I = \{t_{c1}, t_{c2}, \dots t_{cn}\}$. A set of actions to encapsulate (rename to δ) is defined too, in order to force the communication of processes to synchronize with the $Sync_t$ process (1) introduced above: $H = \{t_{s1}, t_{p1}, t_{s2}, t_{p2}, \dots, t_{sn}, t_{pn}\}$. The process is then composed (using the parallel composition operator of the ACP) with the rest of the system (2).

Taking into account the case, when a transition without pre-places occurs within the net structure (Figure 3 b)), the following solution is proposed: for every such transition t, a new pre-place is created, which is connected to the transition by two arcs, such that the firing properties of the transition are preserved (Figure 3 c)). Combining the basic principles explored so far, we are able to construct the terms for more complicated net structures.

Definition 3. Let the Petri net be given by the $N = (P, T, pre, post)$, $m \in \mathbb{N}^k$ defines the initial marking and $k = |P|$. Then the algebraic (ACP) semantics for the Petri net N and the marking m is given by the formula:

$$\mathcal{A}(N, m) = \tau_I(\partial_H(E(p_1)^{(i_1)} \| \dots \| E(p_k)^{(i_k)} \| Sync_{t1} \| \dots \| Sync_{ts})). \tag{2}$$

Within the Equation (2), $E(p_i)$ stands for an ACP-term, defined according to a Petri net structure in the vicinity of the place p_i (see Definition 2). The value i_j, given by $i_j = m(p_j)$, represents the marking with respect to the place $p_j, 1 \leq j \leq k$. By the $E(p_j)^{(i)}$ we mean the term $E(p_j) \| \dots \| E(p_j)$, representing a multiple (i-times) parallel composition of a process $E(p_j)$. Note that the $E(p_j)^{(0)} = \delta$. The $Sync_{ti}$ $(1 \leq i \leq s)$ processes are composed to the rest of the system in the case there are s synchronizing transitions within the source (Petri net) specification.

The size of a process $\mathcal{A}(N, m)$, corresponding to Petri net N with the marking m, created by the transformation depends on the number of marked places in the marking m $(E(p_1) \| \dots \| E(p_k))$, as well as on the number of synchronizing transitions within the net N $(Sync_{t1} \| \dots \| Sync_{ts})$. If we suppose the number of tokens in the marking m is $O(k)$, where the $k = |P|$ and the number of synchronizing transitions s is $O(n)$, where $n = |T|$, we can express the number of process variables in the term $\mathcal{A}(N, m)$ by the sum $O(k) + O(n)$. However, the complexity of the term can vary in the case we start to modify it in order to investigate the behavior of the system it describes, as it can be observed in an example introduced within the next section.

Theorem 1. For given Petri net $N = (P, T, pre, post)$, m, m' markings of the net N, ACP-terms $\mathcal{A}(N, m)$ and $\mathcal{A}(N, m')$, representing the algebraic semantics for the net N, and transition $t \in T$ holds:

$$(N, m) \xrightarrow{t} (N, m') \Rightarrow \mathcal{A}(N, m) \xrightarrow{t} \mathcal{A}(N, m').$$

Proof. The proof is constructed by examining all the elementary cases given by the structure of a Petri net from the point of view of its individual places, as described in the Definition 2. If the step in computation of the Petri net N exists $(N, m) \xrightarrow{t} (N, m')$, so the transition t is enabled in the marking m ($\forall p_i \in ({}^\bullet t) : m(p_i) \geq pre(p_i, t)$) and after firing it produces the new marking m' ($\forall p_i \in P : m'(p_i) = m(p_i) - pre(p_i, t) + post(p_i, t)$), a step should also exist in the corresponding algebraic specification $\mathcal{A}(N, m) \xrightarrow{t} \mathcal{A}(N, m')$. As it was defined in the Definition 3, the algebraic semantics for the Petri net N with the marking m is given by:

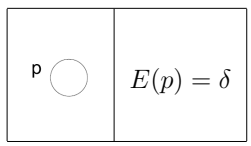
$$\mathcal{A}(N, m) = \tau_I(\partial_H(E(p_1)^{(i_1)} \| \ldots \| E(p_k)^{(i_k)} \| Sync_{t1} \| \ldots \| Sync_{ts})). \tag{3}$$

According to transition relations of ACP (Table 2), a step within the algebraic specification can be performed either by executing an atomic action of a process, or by means of communication of two processes, when the corresponding communication function is defined.

Within the Petri net N, the type of the transition t under consideration can be classified according to size of the set of its pre-places $|{}^\bullet t|$. The case $|{}^\bullet t| = 1$ applies to all the following elementary situations of the Definition 2, except the case 1 (the deadlock) and the case 5 (synchronization, where $|{}^\bullet t| \geq 2$), which will be treated in a slightly different way within the proof. Let us suppose ${}^\bullet t = \{p\}$ (except the cases 1 and 5) and $m(p) \geq pre(p, t)$ (where $m(p)$ is the marking of the place p in the Petri net N), so the transition t is enabled and can fire. Then it also holds that $m'(p_i) = m(p_i) + post(p_i, t) - pre(p_i, t), p_i \in P$ is a new marking of N after firing the transition t. If $E(p)$ represents the corresponding ACP semantics for the process initiated in the place p (Definition 2), then the step (t) is enabled in the $E(p)$ and the $E(p)$ is present in the $\mathcal{A}(N, m)$ (Definition 3).

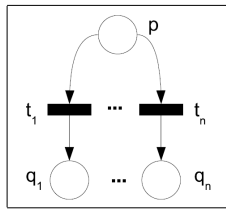
In the following lines we will explore the elementary cases in the order given by the Definition 2.

1. The case represents the situation, where no computation is available, since no transition is connected to the place p.



   The corresponding algebraic term (δ), representing a deadlock is generated.

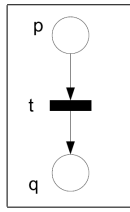
2. The case represents the alternative composition, where the corresponding algebraic term is generated in a way:



$$E(p) = t_1 \cdot E(q_1) + t_2 \cdot E(q_2) + \ldots + t_n \cdot E(q_n)$$

According to transition relations of process algebra ACP (Table 2, case c) the step is possible and one of the alternatives is chosen, given by the transition under consideration ($t \in \{t_1, \ldots, t_n\}$).

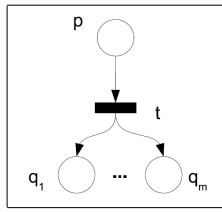
3. The case represents the sequential composition, where the corresponding algebraic term is generated:



$$E(p) = t \cdot E(q)$$

According to the transition relations of process algebra ACP (Table 2, case c), the step t is possible within the corresponding algebraic term.

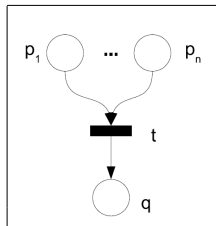
4. The case represents the parallel composition, where the following algebraic term is generated:



$$E(p) = t \cdot (E(q_1) \parallel \ldots \parallel E(q_m))$$

Again, the realization of action t is possible (Table 2, case c) within the corresponding term, regardless the more complicated nature of the process following the action.

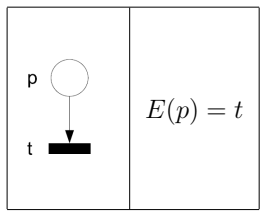
5. In the case, a firing of the transition t with two (or more) preconditions in Petri net N occurs ($|{}^\bullet t| \geq 2$). The situation is depicted in the figure below, which essentially represents the synchronization of processes $p_1, \ldots, p_n$.



$$E(p_1) = t_{p1}, E(p_2) = t_{p2}, \ldots, E(p_n) = t_{pn}$$
$$Sync_t = (t_{s1} \| t_{s2} \| \ldots \| t_{sn}) \cdot t \cdot (E(q) \| Sync_t)$$

If within the marking m of the Petri net N the places $p_1, \ldots, p_n$ contain enough tokens (so the conditions hold: $m(p_1) \geq pre(p_1, t), \ldots, m(p_n) \geq pre(p_n, t)$), the transition t can fire. From the definition of ACP semantics for the Petri net N (Definition 3) and the definition of terms bound to corresponding E-variables (Definition 2, case 5), there is a step available in $\mathcal{A}(N, m)$, too. The step is represented by the communications $(t_{si}|t_{pi} = t_{ci},\ 1 \leq i \leq n)$ of atomic actions $(t_{pi}, 1 \leq i \leq n)$ of synchronizing processes with the corresponding actions $(t_{si}, 1 \leq i \leq n)$ of the $Sync_t$ process. The actions resulting from the communications $(t_{ci}, 1 \leq i \leq n)$ are renamed to internal actions (τ) by means of the operator τ_I and must precede the execution of the action t. Attempts to execute the actions mentioned above (t_{pi} and t_{si}, $1 \leq i \leq n$) without communication are blocked, using the operator ∂_H (Definition 3).

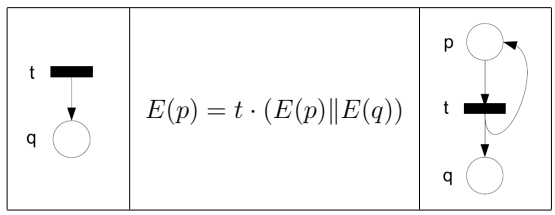
6. Within the case, represented by a transition without post-place(s), the corresponding algebraic term is generated:



$$E(p) = t$$

In this particular case, the step t is possible within the process $E(p)$ in accordance with the transition relations of process algebra ACP (Table 2, case b) and the process terminates after executing the action.

7. The case represents a transition without pre-place(s), where the following algebraic term is generated:



By the substitution introduced within the Definition 2, we have a fragment of Petri net, with the additional place p, which is connected to the transition t by

two arcs, such that the firing properties of the transition are preserved (i.e. if the place p is marked, the transition t can fire arbitrary many times). The step t is possible within the process $E(p)$ in accordance with the transition relations of process algebra ACP (Table 2, case c) and the action t can be executed arbitrary many times.

When the step $\mathcal{A}(N,m) \xrightarrow{t} \mathcal{A}(N,m')$ occurs, the corresponding ACP semantics of the net (N,m') will contain a subterm given by the right hand side of the particular equation for the $E(p)$. The subterm itself represents the particular changes to marking of post-places of the transition t ($t^\bullet$) of the Petri net N. Disappearing of the subterm $E(p)$ on the other hand represents the consumption of the token from the pre-place (p) of the transition t.

We can conclude, that if a step in the Petri net N with the marking m is enabled, so it is enabled also in the corresponding algebraic representation $\mathcal{A}(N,m)$. □

5 AN EXAMPLE

An example is provided within this section to demonstrate the way of using the transformation rules proposed above. The Petri net (depicted in Figure 4) represents a simple system with two processes synchronizing their executions on two common actions. The synchronizing actions are modeled by the transitions t_2 and t_3.

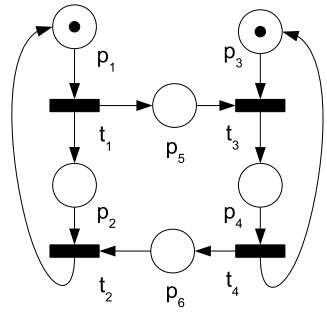


Figure 4. Petri net for two processes

At the beginning we assign the ACP-terms to variables constructed for every place of (source) Petri net N, according to the structure of the net in the neighborhood of the corresponding place.

$$\begin{aligned} &E(p_1) = t_1 \cdot (E(p_5) \| E(p_2)), E(p_2) = t_{2p2}, E(p_3) = t_{3p3}, \\ &E(p_4) = t_4 \cdot (E(p_6) \| E(p_3)), E(p_5) = t_{3p5}, E(p_6) = t_{2p6}. \end{aligned} \tag{4}$$

Next, the synchronizing processes are defined (because of the presence of synchronizing transitions t_2 and t_3):

$$\begin{aligned} Sync_{t2} &= (t_{2s2} \| t_{2s6}) \cdot t_2 \cdot (E(P_1) \| Sync_{t2}), \\ Sync_{t3} &= (t_{3s3} \| t_{3s5}) \cdot t_3 \cdot (E(P_4) \| Sync_{t3}). \end{aligned} \tag{5}$$

We also define four communication functions as we proposed in the Definition 2:

$$t_{2s2}|t_{2p2} = t_{2c2}, t_{3s3}|t_{3p3} = t_{3c3}, t_{2s6}|t_{2p6} = t_{2c6}, t_{3s5}|t_{3p5} = t_{3c5}. \tag{6}$$

A set of actions to encapsulate is defined, in order to force the communication with the $Sync_{t2}$ and the $Sync_{t3}$ processes:

$$H = \{t_{2s2}, t_{2p2}, t_{2s6}, t_{2p6}, t_{3s3}, t_{3p3}, t_{3s5}, t_{3p5}\},$$

as well as a set of actions to abstract away and so hide the internal behavior of the system:

$$I = \{t_{2c2}, t_{2c6}, t_{3c3}, t_{3c5}\}.$$

Since the initial marking of the Petri net N is given as $m_0 = (1, 0, 1, 0, 0, 0)$, only two places (p_1 and p_3) hold tokens. Now initially, only the variables corresponding to these places will be present in an equation describing the algebraic semantics of the Petri net N, together with the processes $Sync_{t2}$ and $Sync_{t3}$, composed by the means of parallel composition. Let the name of the whole composition be Sys and the names of variables $E(p_i)$ be E_i $(1 \leq i \leq 6)$ for the sake of simplicity within this example.

$$\mathcal{A}(N, m_0) = Sys = \tau_I(\partial_H(E_1 \| E_3 \| Sync_{t2} \| Sync_{t3})). \tag{7}$$

According to Equations (4) and (5), we can substitute for the variables present in the specification:

$$Sys = \tau_I(\partial_H(\underline{t_1} \cdot (\underline{E_2} \| \underline{E_5}) \| t_{3p3} \| (t_{2s2} \| t_{2s6}) \cdot t_2 \cdot (E_1 \| Sync_{t2}) \| (t_{3s3} \| t_{3s5}) \cdot t_3 \cdot (E_4 \| Sync_{t3}))).$$

Since the actions $t_{3p3}, t_{2s2}, t_{2s6}, t_{3s3}, t_{3s5}$ are encapsulated by ∂_H, the only available action for execution is the t_1. We also substitute for the E_2 and E_5 process variables and we have (the transition t_1 and the variables E_2 and E_5 are underlined in the equation above for the sake of lucidity):

$$Sys = t_1 \cdot \tau_I(\partial_H(t_{2p2} \| \underline{t_{3p5}} \| \underline{t_{3p3}} \| (t_{2s2} \| t_{2s6}) \cdot t_2 \cdot (E_1 \| Sync_{t2}) \| (\underline{t_{3s3}} \| \underline{t_{3s5}}) \cdot t_3 \cdot (E_4 \| Sync_{t3}))).$$

In this situation all the initial actions of the composed processes are blocked by the ∂_H operation, so the communication (according to definition of communication functions (6)) is the only available action. So we allow to communicate the actions t_{3p5} with t_{3s5} and t_{3p3} with t_{3s3}:

$$Sys = t_1 \cdot \tau_I(\partial_H(t_{2p2} \| (t_{2s2} \| t_{2s6}) \cdot t_2 \cdot (E_1 \| Sync_{t2}) \| (\underline{t_{3c3}} \| \underline{t_{3c5}}) \cdot t_3 \cdot (E_4 \| Sync_{t3}))).$$

After hiding the products of communications (t_{3c3} and t_{3c5}) by renaming them to silent actions τ by means of the abstraction operation (τ_I), we have:

$$Sys = \underline{t_1 \cdot \tau \cdot \tau} \cdot \tau_I(\partial_H(t_{2p2} \| (t_{2s2} \| t_{2s6}) \cdot t_2 \cdot (E_1 \| Sync_{t2}) \| \underline{t_3} \cdot (\underline{E_4} \| Sync_{t3}))).$$

At this point we have the action t_3 which is not affected by the operators τ_I and ∂_H, so it can be placed before them. We also can remove the silent actions (τ) using the axiom B1 (Table 1) and substitute for the variable E_4.

$$Sys = t_1 \cdot t_3 \cdot \tau_I(\partial_H(t_{2p2} \| (t_{2s2} \| t_{2s6}) \cdot t_2 \cdot (E_1 \| Sync_{t2}) \| \underline{t_4} \cdot (\underline{E_6} \| E_3) \| Sync_{t3})).$$

Further, the action t_4 can be moved to the position after the t_3, since it is not affected by the operators τ_I and ∂_H. We also substitute the action t_{2p6} for the variable E_6 within this step.

$$Sys = t_1 \cdot t_3 \cdot t_4 \cdot \tau_I(\partial_H(\underline{t_{2p2}} \| (\underline{t_{2s2}} \| \underline{t_{2s6}}) \cdot t_2 \cdot (E_1 \| Sync_{t2}) \| \underline{t_{2p6}} \| E_3 \| Sync_{t3})).$$

Communications in this case produce the actions t_{2c2} and t_{2c6}.

$$Sys = t_1 \cdot t_3 \cdot t_4 \cdot \tau_I(\partial_H((\underline{t_{2c2}} \| \underline{t_{2c6}}) \cdot t_2 \cdot (E_1 \| Sync_{t2}) \| E_3 \| Sync_{t3})).$$

These actions are not blocked by ∂_H, but are renamed by τ_I to silent actions τ.

$$Sys = t_1 \cdot t_3 \cdot \underline{t_4 \cdot \tau \cdot \tau} \cdot \tau_I(\partial_H(\underline{t_2} \cdot (E_1 \| Sync_{t2}) \| E_3 \| Sync_{t3})).$$

The silent actions can be removed using the axiom B1 (Table 1) and the action t_2 is moved out of scope of the operators τ_I and ∂_H.

$$Sys = t_1 \cdot t_3 \cdot t_4 \cdot t_2 \cdot \tau_I(\partial_H(E_1 \| Sync_{t2} \| E_3 \| Sync_{t3})).$$

After exchanging the order of process variables $Sync_{t2}$ and E_3 we have the equation:

$$Sys = t_1 \cdot t_3 \cdot t_4 \cdot t_2 \cdot \tau_I(\partial_H(E_1 \| E_3 \| Sync_{t2} \| Sync_{t3})).$$

Where the term $\tau_I(\partial_H(E_1 \| E_3 \| Sync_{t2} \| Sync_{t3}))$ is indeed the one from which we started the derivation (7), so we have:

$$Sys = t_1 \cdot t_3 \cdot t_4 \cdot t_2 \cdot Sys. \tag{8}$$

The prefix $t_1 \cdot t_3 \cdot t_4 \cdot t_2$ represents the trace of actions executed by the process Sys before it starts the execution from the initial state again. Thus finally, the operation of the system can be described by the following expression, where the symbol ω expresses the arbitrary number of repetitions.

$$Sys = (t_1 \cdot t_3 \cdot t_4 \cdot t_2)^{\omega}. \tag{9}$$

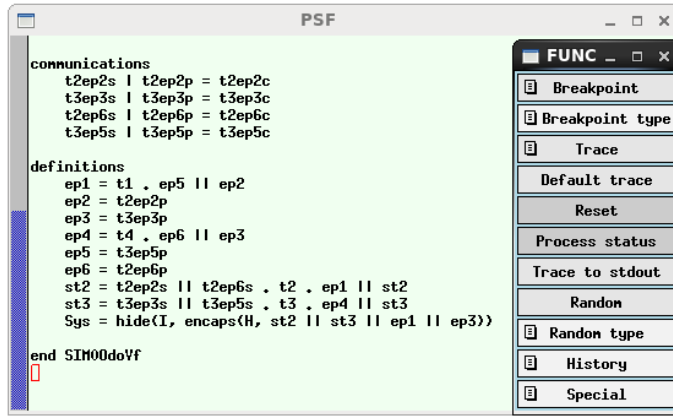


Figure 5. PSF-Toolkit simulation environment

For the purpose of practical validation of the results obtained by means of manual derivation given above, we also performed the simulation using the PSF-Toolkit [14]. The environment of the PSF-Toolkit, together with (part of) the specification is depicted in Figure 5.

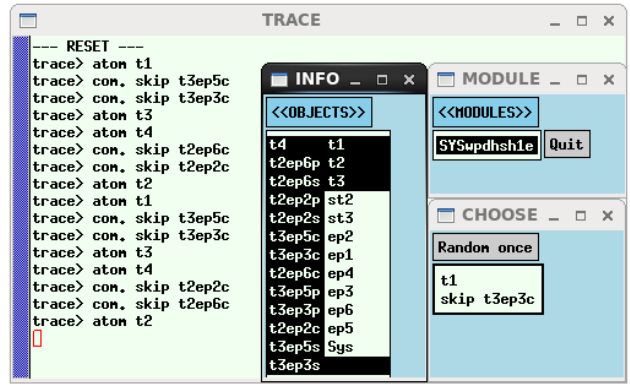


Figure 6. Simulation of the system using the PSF-Toolkit

The names of processes and actions here differ slightly in some cases (like the $st2$ instead of $Sync_{t2}$, or the $t2ep6c$ instead of t_{2c6}), but we hope they are still clear enough. The simulation itself provides the results, which are in-line with the results stated above and the trace of actions (Figure 6) acquired by the simulation of the system corresponds to (9).

Further, we can define a new process (named *ReSys* in Figure 7), corresponding to the external behavior of the system, as it is stated above (8). Finally, we can compare the behavior of the two processes (*Sys* and *ReSys*), using the

```
definitions
  ep1 = t1 . (ep5||ep2)
  ep2 = t2ep2p
  ep3 = t3ep3p
  ep4 = t4 . (ep6||ep3)
  ep5 = t3ep5p
  ep6 = t2ep6p
  st2 = (t2ep2s||t2ep6s) . t2 . (ep1||st2)
  st3 = (t3ep3s||t3ep5s) . t3 . (ep4||st3)
  Sys = hide(I,encaps(H,(st2||st3||ep1||ep3)))
  ReSys = t1 . t3 . t4 . t2 . ReSys
```

Figure 7. Process definitions with the *ReSys* process

equiv tool of the PSF-Toolkit (Figure 8) and find that the processes are bisimilar.

```
[tom@localhost 2018b]$ psf wpd_hsh1er.psf
[tom@localhost 2018b]$ trans SYSwpdhsh1er.til SYSwpdhsh1er.out
[tom@localhost 2018b]$ equiv SYSwpdhsh1er.out

Enter state #1 (none to exit): Sys
Enter state #2 (none to exit): ReSys
Sys and ReSys are branching bisimilar
```

Figure 8. Test of equivalence

6 CONCLUSION

In the paper we presented an approach for constructing an algebraic semantics for Petri nets, based on Algebra of Communicating Processes (ACP). A variable is created for every place of the given Petri net and a term is bound to the variable, which expresses the process initiated in the place. The description of a process representing the computations of Petri net is given by the parallel composition of all the variables associated with the places holding tokens within the initial marking.

The ideas presented in the paper were implemented within the Petri2ACP tool. Similarly, as it was in the case of the Petri2APC tool [31], where the first step is parsing the PNML specification of a Petri net, the PNML Framework [19] was used in this case, too. The resulting ACP specification is produced in a format of the PSF-Toolkit. The tool was implemented using Java platform and is intended to perform a deeper exploration of properties of the transformation, as well as to enable its practical utilization.

As it was demonstrated by the example, a manual investigation of system behavior is possible, but it can be complicated and error-prone task. So having the tool support (e.g. the PSF-Toolkit in this case) available to help with this task is very useful and it was indeed one of the main sources of motivation for this work. One of our previously developed transformations (Petri2APC [30]), which utilizes a dedicated process algebra APC (Algebra of Process Components), provides simpler algebraic specifications indeed, but there is currently no tool support for an-

alyzing such specifications available. The algebraic specifications produced by the Petri2APC transformation are often simpler, since in the APC special constructs are available for expressing the synchronization of processes. So a certain price to pay for the possibility of utilizing powerful tools in the case of ACP is the more complicated modeling of process synchronization based on communication, encapsulation and abstraction.

Although a reverse transformation to the one presented in this work, i.e. the Petri net semantics for ACP terms, is not covered by this paper in detail; we published some results in this respect in the past [28, 29]. An interesting application area of that transformation and the corresponding tool (ACP2Petri) is the design and analysis of communication protocols based on integration of the two mentioned formal methods [32]. It would be very interesting to study the relations between the two transformations in more details.

In the future, it would also be interesting to relate a high-level Petri nets with some variant of data-enriched process algebra [8]. Another possible direction of the future research is connected with the process algebra APC and development of tools supporting the analysis of APC specifications. It would also be useful to enhance the implementation of the transformation tool Petri2ACP in order to include the support for additional tools, like mCRL2 [12].

Acknowledgment

This work was supported by VEGA Grant No. 1/0341/13 Principles and methods of automated abstraction of computer languages and software development based on the semantic enrichment caused by communication.

REFERENCES

[1] Astesiano, E.—Broy, M.—Reggio, G.: Algebraic Specification of Concurrent Systems. In: Astesiano, E., Kreowski, H. J., Krieg-Brückner, B. (Eds.): Algebraic Foundations of Systems Specification. Springer, Berlin, Heidelberg, IFIP State-of-the-Art Reports, 1999, pp. 467–520, doi: 10.1007/978-3-642-59851-7_13.

[2] Baeten, J. C. M.—Weijland, W. P.: Process Algebra. Cambridge University Press, 1990, doi: 10.1017/cbo9780511624193.

[3] Baeten, J. C. M.: A Brief History of Process Algebra. Theoretical Computer Science, Vol. 335, 2005, No. 2–3, pp. 131–146, doi: 10.1016/j.tcs.2004.07.036.

[4] Baeten, J. C. M.—Basten, T.: Partial-Order Process Algebra (and Its Relation to Petri Nets). In: Bergestra, J. A., Ponse, A., Smolka, S. A. (Eds.): Handbook of Process Algebra. Chapter 13. Elsevier Science, Amsterdam, The Netherlands, 2001, pp. 769–872.

[5] Baldan, P.—Bonchi, F.—Gadducci, F.—Monreale, G. V.: Asynchronous Traces and Open Petri Nets. In: Bodei, C., Ferrari, G., Priami, C. (Eds.): Programming Languages with Applications to Biology and Security. Springer, Cham, Lecture

Notes in Computer Science, Vol. 9465, 2015, pp. 86–102, doi: 10.1007/978-3-319-25527-9_8.

[6] Basten, T.—Voorhoeve, M.: An Algebraic Semantics for Hierarchical P/T Nets. Computing Science Report, Ph.D. thesis, Eindhoven University of Technology, 1995, doi: 10.1007/3-540-60029-9_33.

[7] Basten, T.: In Terms of Nets: System Design With Petri Nets and Process Algebra. Ph.D. thesis, Eindhoven University of Technology, 1998, doi: 10.6100/IR516117.

[8] Bergstra, J. A.—Middelburg, C. A.: Data Linkage Algebra, Data Linkage Dynamics, and Priority Rewriting. Fundamenta Informaticae, Vol. 128, 2013, No. 4, pp. 367–412.

[9] Bernardinello, L.—De Cindio, F.: A Survey of Basic Net Models and Modular Net Classes. In: Rozenberg, G. (Ed.): Advances in Petri Nets 1992. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 609, 1992, pp. 304–351, doi: 10.1007/3-540-55610-9_177.

[10] Best, E.—Wimmel, H.: Structure Theory of Petri Nets. In: Jensen, K., van der Aalst, W. M. P., Balbo, G., Koutny, M., Wolf, K. (Eds.): Transactions on Petri Nets and Other Models of Concurrency VII. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, 2013, Vol. 7480, pp. 162–224, doi: 10.1007/978-3-642-38143-0_5.

[11] Best, E.—Devillers, R.—Koutny, M.: Petri Nets, Process Algebras and Concurrent Programming Languages. Lectures on Petri Nets II: Applications (ACPN 1996). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1492, 1998, pp. 1–84, doi: 10.1007/3-540-65307-4_46.

[12] Cranen, S.—Groote, J. F.—Keiren, J. J. A.—Stappers, F. P. M.—de Vink, E. P.—Wesselink, J. W.—Willemse, T. A. C.: An Overview of the mCRL2 Toolset and Its Recent Advances. In: Piterman, N., Smolka, S. A. (Eds.): Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 7795, 2013, pp. 199–213, doi: 10.1007/978-3-642-36742-7_15.

[13] Desel, J.—Juhás, G.—Lorenz, R.: Process Semantics of Petri Nets over Partial Algebra. In: Nielsen, M., Simpson, D. (Eds.): Application and Theory of Petri Nets 2000 (ICATPN 2000). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 1825, 2000, pp. 146–165, doi: 10.1007/3-540-44988-4_10.

[14] Diertens, B.: Software Engineering with Process Algebra. Ph.D. thesis, University of Amsterdam, 2009.

[15] Ding, Z.—Liu, J.—Wang, J.: An Executable Service Composition Code Automatic Creation Tool Based on Petri Net Model. Computing and Informatics, Vol. 32, 2013, No. 5, pp. 968–986.

[16] Fokkink, W.: Modelling Distributed Systems. Protocol Verification with mCRL. 2nd edition. Springer, 2011.

[17] Gorrieri, R.—Versari, C.: A Process Calculus for Expressing Finite Place/Transition Petri Nets. Proceedings of 17th International Workshop on Expressiveness in Concurrency (EXPRESS'10), Paris, France, August 30, 2010, doi: 10.4204/eptcs.41.6.

[18] Gorrieri, R.: Language Representability of Finite P/T Nets. In: Bodei, C., Ferrari, G., Priami, C. (Eds.): Programming Languages with Applications to Biology and Security. Springer, Cham, Lecture Notes in Computer Science, Vol. 9465, 2015, pp. 262–282, doi: 10.1007/978-3-319-25527-9_17.

[19] Hillah, L. M.—Kordon, F.—Petrucci, L.—Trèves, N.: PNML Framework: An Extendable Reference Implementation of the Petri Net Markup Language. In: Lilius, J., Penczek, W. (Eds.): Applications and Theory of Petri Nets (PETRI NETS 2010). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6128, pp. 318–327, doi: 10.1007/978-3-642-13675-7_20.

[20] Hudák, Š.: Reachability Analysis of Systems Based on Petri Nets. Elfa s.r.o., Košice, 272 pp., 1999.

[21] Hudák, Š.—Zaitsev, D. A.—Korečko, Š.—Šimoňák, S.: MFDTE/PNtool – A Tool for the Rigorous Design, Analysis and Development of Concurrent and Time-Critical Systems. Acta Electrotechnica et Informatica, Vol. 7, 2007, No. 4, pp. 5–12.

[22] Lodaya, K.: A Regular Viewpoint on Processes and Algebra. Acta Cybernetica, Vol. 17, 2006, No. 4, pp. 751–763.

[23] Milner, R.: A Calculus of Communicating Systems. Lecture Notes in Computer Science, Vol. 92, Springer, Berlin, 1980.

[24] Murata, T.: Petri Nets: Properties, Analysis and Applications. Proceedings of the IEEE, Vol. 77, 1989, No. 4, pp. 541–580, doi: 10.1109/5.24143.

[25] Olderog, E. R.: Nets, Terms and Formulas. Cambridge University Press, 1991, doi: 10.1017/cbo9780511526589.

[26] Paige, R. F.: Formal Method Integration via Heterogeneous Notations. Ph.D. dissertation, University of Toronto, 1997.

[27] Sobociński, P.: Representations of Petri Net Interactions. In: Gastin, P., Laroussinie, F. (Eds.): CONCUR 2010 – Concurrency Theory. Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 6269, 2010, pp. 554–568, doi: 10.1007/978-3-642-15375-4_38.

[28] Šimoňák, S.—Hudák, Š.—Korečko, Š.: ACP2PETRI: A Tool for FDT Integration Support. Proceedings of 8th International Conference (EMES '05), Oradea, Felix Spa, Romania, 2005, pp. 122–127.

[29] Šimoňák, S.: Formal Methods Transformation Optimizations within the ACP2PETRI Tool. Acta Electrotechnica et Informatica, Vol. 6, 2006, No. 1, pp. 75–80.

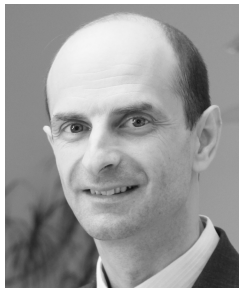
[30] Šimoňák, S.—Hudák, Š.—Korečko, Š.: APC Semantics for Petri Nets. Informatica, Vol. 32, 2008, No. 3, pp. 253–260.

[31] Šimoňák, S.—Hudák, Š.—Korečko, Š.: PETRI2APC: Towards Unifying Petri Nets with Other Formal Methods. Proceedings of the Seventh International Scientific Conference Electronic Computers and Informatics (ECI 2006), Herľany, Slovakia, pp. 140–144.

[32] Šimoňák, S.: Verification of Communication Protocols Based on Formal Methods Integration. Acta Polytechnica Hungarica, Vol. 9, 2012, No. 4, pp. 117–128.

[33] Zhang, X.—Yao, S.: Bank Switching Performance Verification with Object-Oriented Timed Petri Nets. Proceedings of the 2013 IEEE 8th Conference on In-

dustrial Electronics and Applications (ICIEA 2013), Melbourne, Australia, 2013, pp. 1664–1669, doi: 10.1109/iciea.2013.6566636.



Slavomír Šimoňák received his M.Sc. degree in computer science in 1998 and his Ph. D. degree in computer tools and systems in 2004, both from the Technical University of Košice, Slovakia. He is currently Associate Professor at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice, Slovakia. His research interests include formal methods integration and application, communication protocols, algorithms, and data structures.

Martin Tomášek received his M.Sc. degree in computer science in 1998 and his Ph.D. degree in software and information systems in 2005 both at the Faculty of Electrical Engineering and Informatics of the Technical University of Košice, Slovakia. Currently he is Associate Professor at the Department of Computers and Informatics of the Faculty of Electrical Engineering and Informatics at the Technical University of Košice, Slovakia. His research interests include distributed systems, component-based systems, and concurrency theory.