APC Semantics for Petri Nets

Slavomír Šimoňák, Štefan Hudák and Štefan Korečko
Department of Computers and Informatics,
Technical University of Košice, Letná 9, 042 00 Košice, Slovak Republic
E-mail: slavomir.simonak@tuke.sk, stefan.hudak@tuke.sk, stefan.korecko@tuke.sk

Keywords: Petri nets semantics, process algebra, algebra of process components

Received: July 13, 2007

The paper deals with an algebraic semantics for Petri nets, based on a process algebra APC (Algebra of Process Components) by the authors. APC is tailored especially for describing processes in Petri nets. This is done by assigning special variables (called E-variables here) to every place of given Petri net, expressing processes initiated in those places. Algebraic semantics is then given as a parallel composition of all the variables, whose corresponding places hold token(s) within the initial marking. Resulting algebraic specification preserves operational behavior of the original net-based specification.

Povzetek: Članek opisuje algebro semantike za Petri mreže.

1 Introduction

An assertion widely accepted in formal methods community states, that there will never be invented a single formal method, that will cover all aspects of the system in acceptable way [12]. The latter is mainly because of the complexity of the system and vast variety of its features (aspects) to be covered for the system to be modeled and designed. As a consequence of that situation, many formal description techniques (FDTs) exist and are used nowadays. That reflects the fact that one feature f of the system is more readily expressed in FDT (say) F_1, than it is the case for f in F_2. To cope with that situation there have been attempts to integrate two or more formal methods. Main motivation for using FDT in design and analysis of computer-based systems lies in everyday growing dependence of human society on such the systems, particulary those applied in safety-critical domains (such as military weaponry, aircraft transport, medicine, etc.). The situation just described put strong requirements on new methods of the design, analysing and maintaining of such the systems. Time-critical issue of to be able to cope with malicious behaviour of the systems in limited time period, dictates strongly to deal with the problem in an automated way. The latter is impossible without using FDTs in proper combination and integration in the frame of computer-based design and analysis environments. Yet a formal way how to incorporate conditions to guarantee the safety of system designed is an example of another problem which underlines the importance of expolitation of FDTs. Guided by the considerations mentioned an approach has been applied at the home institution of the authors to create an environment for the design and analysis of discrete systems based on integration of three FDTs: Petri Nets (PN), process algebras and B-Method. That is why we have chosen the acronym mFDTE for it (multi FDT Environment). The choice of the FDTs has been motivated by their abilities to cover in some mutually complementary ways a chosen set of system's features. In this work we pay attention to two of the FDTs chosen: PN and process algebras. While on one side PN posses nice properties suited for system modelling (formal and graphical language) and analysis (invariants, reachability), on the other side they suffer from a lack of formally sound and effective methods of their de/composition. The latter can be considered as an essential drawback as far as the modular system design is concerned. Process algebras (CCS, CSP, ACP)[1] on the other hand support composionality, by their definition, so PN and process algebras can be considered to meet the complementarity property in the above sense.

The paper is organized in the following way: Section 1 is introductory one, whole related works are briefly summarized in Section 2. In Section 3, basic notions and definitions for the class of Petri nets used are given. Algebra of Process Components is defined in Section 4. Notion of term is presented as a mean for describing processes, axioms are given and operational semantics is assigned to process expressions (terms). Section 5 concentrates on the algebraic semantics construction for a Petri net given. A special variable is assigned to every place of the Petri net. Construction rules are defined for assigning a term to the variable which represents all the computations which can be initiated at the corresponding place. An example provided in Section 6 demonstrates the approach introduced above. Section 7 concludes the paper and contains a summary of the results and concepts presented.

2 Related work

An active research has been performed in the area of combining Petri nets and process algebras during last years

[4, 5, 2, 6, 10, 11]. It will be mentioned further [3], where authors propose an approach to algebraic semantics for Hierarchical P/T nets. PTNA (Place/Transition Net Algebra) is defined there, based on process algebra ACP and an algebraic semantics for P/T nets is given such that a P/T net and its term representation have the same operational behavior. The actions of the algebra presented correspond to the consumption and production of tokens by transitions. Results achieved are further extended to hierarchical P/T nets. In [11] relations among nets, terms and formulas are treated. Particularly relations are defined via properly defined semantics: net semantics of terms and process semantics of nets. The most influential works in the line we follow here are [5, 2]. In [5] relations between the process algebra, called there PBC (Petri Box Calculus) and a class of Place Transition Nets (safe P/T nets) are studied. Syntax and semantics of PBC terms are carefully selected to allow to define a transformation yielding P/T nets preserving structural operational semantics of the source terms. The transformation allows composition of P/T nets. In work [2] authors treat the issue of partial-order algebras and their relations to P/T nets based on the theory of BPA and ACP.

Within our work we propose the general approach to characterising PN in form of E-(B-) terms. General (not only interleaving) semantics is given, and the results obtained in this respect are published in [9]. mFDT Environment is under construction, based on FDT interfaces by the authors, which aims to integrate the three formal methods mentioned above.

3 Petri Nets

We assume the class of ordinary Petri nets [18] within this paper, and brief description of the basic notions follows [7].

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.*

By the marking of PN $N = (P, T, pre, post)$ we mean a totally defined function m: $P \to \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers. We denote marked net with initial marking m_0 as $N_0 = (N, m_0)$ or $N_0 = (P, T, pre, post, m_0)$.

Some useful notations can be defined:

$$^\bullet t = \{p | pre(p, t) \neq 0\} \text{the set of preconditions of } t$$

$$t^\bullet = \{p | post(p, t) \neq 0\} \text{the set of postconditions of } t$$

$$p^\bullet = \{t | pre(p, t) \neq 0\},\ ^\bullet p = \{t | post(p, t) \neq 0\}$$

We say that 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 a 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 within the paper for expressing a step of computation ($m \xrightarrow{t} m'$) within the Petri net N. The set of reachable markings for given Petri net $N_0 = (P, T, pre, post, m_0)$ we define by

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

where $\sigma = t_1, t_2 ... t_r$ stands for an admissible firing sequence in N_0. We also can define the language of Petri net N_0:

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

4 APC - Algebra of Process Components

Process algebra APC [8, 16] is inspired by the process algebra ACP [1]. ACP is modified in a way, that allows for comfortable description of PN processes in the algebraic way. We use the same operators for the sequential (·) and the alternative (+) composition respectively and corresponding axioms also hold in algebra APC (Table 1). ACP's communication function (γ) and its extension - communication merge operator (|), are not present in APC. A composition function (π) and a special composition operator (|||) are introduced into the algebra APC instead. The impact of an introduction of the two operators will be treated later.

APC is defined as a couple (P, Σ), where P (the domain) is represented by the set of constants, set of variables and set of all processes (terms) we are able to express. Σ (the signature) contains function (operator) symbols. It is supposed that, the set of variables contain arbitrary many of them (x,y, ...). Terms containing variable(s) are called *open terms*, otherwise terms are *closed.*

4.1 Syntactical issues

From the syntactical point of view APC contains a number of constants $a, b, c, ...$ (we use the set $A = \{a, b, c, ...\}$ for referring to them) a special constant δ (deadlock) and operators: +, ·, $\parallel$ (parallel composition), $\lfloor\!\lfloor$ (left merge) and $|||$ (process component composition). It also contains a (partial) commutative composition function π, denoting the merging of process components. Now we will define APC terms:

Definition 2. *1. variables are APC terms,*

2. constants $a \in A$ and a special constant δ are APC terms,

3. if u, v are APC terms, then $u + v, u \cdot v$, $u \parallel v, u \lfloor\!\lfloor v, u|||v$ are APC terms,

4. if u is APC term, then $u^{[c]}$, $c \in \mathbb{N}$ is also APC term.

All these terms are part of P - the domain of APC. P can further be subdivided into two parts P_A and P_C ($P = P_A \cup P_C$). Terms belonging to the set P_A defined by items 1, 2 and 3 of Definition 2 (i.e. those without the superscript notation) represent the set of *true processes*. Terms from the set P_C (superscripted) represent the set of *process components*.

Only difference between the (true) process and process component is, that while the process is able to execute actions, process component is introduced for synchronization purpose only. The latter is only able to join with its counterpart(s) (other process component(s) fitting for being synchronized to) in order to form a true process. The composition function π is defined as follows:

$$\pi : P_C \times ... \times P_C \rightarrow P_A \tag{1}$$

A connection between the composition function π and the process component composition operator ($|||$) can be expressed as: $x_1|||...|||x_n = \pi(x_1, ..., x_n)$ when $\pi(x_1, ..., x_n)$ is defined. Axioms of algebra APC can be found in Table 1. Within the table $u, v, z, x_1, ..., x_n$ stand for processes, $a \in A$ and δ are constants.

$u + v = v + u$	A1
$u + u = u$	A2
$(u + v) + z = u + (v + z)$	A3
$(u + v) \cdot z = uz + vz$	A4
$(u \cdot v) \cdot z = u \cdot (v \cdot z)$	A5
$u\|v = u \lfloor\!\lfloor v + v \lfloor\!\lfloor u + u\|\|\|v$	A6
$(u + v)\|\|\|z = u\|\|\|z + v\|\|\|z$	A7
$u\|\|\|(v + z) = u\|\|\|v + u\|\|\|z$	A8
$au \lfloor\!\lfloor v = a(u\|v)$	A9
$(u + v) \lfloor\!\lfloor z = u \lfloor\!\lfloor z + v \lfloor\!\lfloor z$	A10
$x_1\|\|\|...\|\|\|x_n = \pi(x_1, ..., x_n)$ if $\pi(x_1, ..., x_n)$ is defined $x_1\|\|\|...\|\|\|x_n = \delta$ otherwise	A11
$u + \delta = u$	A12
$\delta \cdot u = \delta$	A13

Table 1: Axioms of APC

Theorem 1. *In the case of the parallel composition of more than two processes the following equality (expansion theorem) can be proven:*

$$x_1\|...\|x_n = \sum_{1\le i\le n} x_i \lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n, j\ne i} x_j) + \tag{2}$$

$$\sum_{2\le k\le n} \sum^{1\le i_1<...<i_k\le n} (x_{i_1}|||...|||x_{i_k}) \lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n, j\ne i_1,...,i_k} x_j)$$

Proof 1. *By induction on n, the number of processes. The case for $n = 2$ is treated by the axiom A6, Table 1. The induction step is as follows:*

$$(x_1\|...\|x_{n+1}) = (x_1\|...\|x_n)\|x_{n+1}$$

Denoting the RHS of original theorem as E, we can write:

$$(x_1\|...\|x_n)\|x_{n+1} = E \lfloor\!\lfloor x_{n+1} + x_{n+1} \lfloor\!\lfloor E + E|||x_{n+1} \tag{3}$$

Now we have three summands, each of them will be treated separately. Let's start dealing with the first of them.

$$E \lfloor\!\lfloor x_{n+1} = (\sum_{1\le i\le n} x_i \lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n, j\ne i} x_j)) \lfloor\!\lfloor x_{n+1} +$$

$$(\sum_{2\le k\le n} \sum^{1\le i_1<...<i_k\le n} (x_{i_1}|||...|||x_{i_k})$$

$$\lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n, j\ne i_1,...,i_k} x_j)) \lfloor\!\lfloor x_{n+1} =$$

$$\sum_{1\le i\le n} x_i \lfloor\!\lfloor ((\mathop{\|}\limits^{1\le j\le n, j\ne i} x_j) \lfloor\!\lfloor x_{n+1}) +$$

$$\sum_{2\le k\le n} \sum^{1\le i_1<...<i_k\le n} (x_{i_1}|||...|||x_{i_k})$$

$$\lfloor\!\lfloor ((\mathop{\|}\limits^{1\le j\le n, j\ne i_1,...,i_k} x_j) \lfloor\!\lfloor x_{n+1}) =$$

$$\sum_{1\le i\le n} x_i \lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n+1, j\ne i} x_j) +$$

$$\sum_{2\le k\le n} \sum^{1\le i_1<...<i_k\le n} (x_{i_1}|||...|||x_{i_k})$$

$$\lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n+1, j\ne i_1,...,i_k} x_j)$$

The second summand of (3) can be expressed as follows:

$$x_{n+1} \lfloor\!\lfloor E = x_{n+1} \lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n+1, j\ne n+1} x_j)$$

The third one represents the process components composition:

$$E|||x_{n+1} = (\sum_{1\le i\le n} x_i \lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n, j\ne i} x_j))|||x_{n+1} +$$

$$(\sum_{2\le k\le n} \sum^{1\le i_1<...<i_k\le n} (x_{i_1}|||...|||x_{i_k})$$

$$\lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n, j\ne i_1,...,i_k} x_j))|||x_{n+1} =$$

$$\sum_{1\le i\le n} (x_i|||x_{n+1}) \lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n, j\ne i} x_j) +$$

$$(\sum_{3\le k\le n+1} \sum^{1\le i_1<...<i_k\le n+1} (x_{i_1}|||...|||x_{i_k})$$

$$\lfloor\!\lfloor (\mathop{\|}\limits^{1\le j\le n, j\ne i_1,...,i_k} x_j))$$

Summing up the three summands we have:

$$(x_1\|...\|x_n\|x_{n+1}) = \sum_{1\leq i\leq n+1} x_i \mathbin{|\!\!\lfloor}(\mathop{\|}\limits^{1\leq j\leq n+1, j\neq i} x_j) +$$

$$\sum_{2\leq k\leq n+1} \sum^{1\leq i_1<...<i_k\leq n+1} (x_{i_1}|||...|||x_{i_k})$$

$$\mathbin{|\!\!\lfloor}(\mathop{\|}\limits^{1\leq j\leq n+1, j\neq i_1,...,i_k} x_j)$$

$\square$

4.2 Semantics issues

Constants $a, b, c \in A$ are called atomic actions, and are considered indivisible actions (events). The sequential composition operator $(\cdot)$ function can be explained as follows: $x{\cdot}y$ is the process that first executes x and after finishing it, starts y. The alternative composition $(+)$: $x{+}y$ is the process that either executes x or y (choice). The meaning of parallel composition $(\|)$ follows: considering the merge of two processes $x\|y$, we recognize three possibilities to proceed. Either we start with a first step of x (given by $x \mathbin{|\!\!\lfloor} y$), or a first step from y ($y \mathbin{|\!\!\lfloor} x$) or we check a possibility to compose processes by means of process components composition operator - $x|||y$. The result of this composition of course is different from δ only in a case, when the composition function π is defined.

To assign an operational semantics to process expressions, we determine, which actions a process can perform. The fact, that process represented by the term t can execute 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 a fact that t can terminate by executing a. An inductive definition of action relations is given in the Table 2.

$a \xrightarrow{a} \surd$	
$u \xrightarrow{a} u' \Rightarrow$	$u + v \xrightarrow{a} u'$
	$v + u \xrightarrow{a} u'$
	$u \cdot v \xrightarrow{a} u' \cdot v$
	$u\|v \xrightarrow{a} u'\|v$
	$v\|u \xrightarrow{a} v\|u'$
	$u \mathbin{\|\!\!\lfloor} v \xrightarrow{a} u'\|v$
$u \xrightarrow{a} u'$, $\pi(u^{[1]}, ..., u^{[n]}) = u \Rightarrow u^{[1]}\|\|\|...\|\|\|u^{[n]} \xrightarrow{a} u'$	
$a \in A,\ u \in P_A,\ u^{[1]}, ..., u^{[n]} \in P_c,\ u', v \in P$	

Table 2: Transition relations for APC terms

5 APC semantics for Petri Nets

In this section the transformation description is given in detail [16]. We start with creating a special variable for every place in the PN N to be transformed. We call these variables *E-variables* here, and they will be bound to terms, representing possible computations started from given place in PN, later. So the value (term) assigned to a particular variable depends on the structure of the net in the vicinity of a place associated. So considering the place p, variable $E(p)$ will be bound to a term representing all the computations within the net N, which are initiated in p.

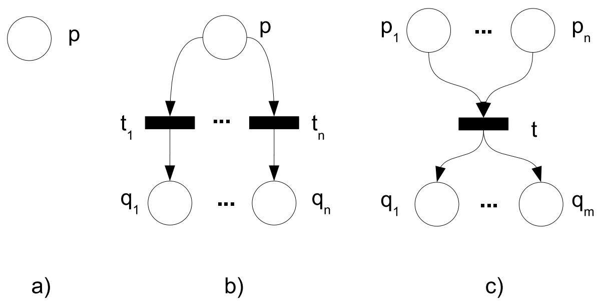

Figure 1: Petri net fragments

Basic situations are captured in Fig. 1. In the case a) a situation is depicted, where no arcs are connected to the place investigated.

This results to the assignment of a term representing no computations to the variable corresponding to such place, i.e. δ (deadlock). Case b) stands for an alternative composition (choice). If a token is situated in place p, a choice is to be made, and only one of transitions $t_1, ..., t_n$ can fire. Case c) represents general composition, where tokens must be present in all pre-places of transition t. If some of these places does not contain a token, firing of t is not possible. After firing of t, however post-places of it are marked and thus processes initiated in those places are enabled.

General composition (case c) can be understood as a generalization of the three basic compositions - sequential, parallel and synchronization (Fig. 2) - three of four basic composition mechanisms (with alternative composition) used within the APC. If n is the number of pre-places and m, the number of post-places of a transition t:

– $n = 1 \wedge m = 1$ we obtain sequential composition (case a) of Fig. 2),

– $n = 1 \wedge m > 1$ we obtain parallel composition (case b) of Fig. 2),

– $n > 1 \wedge m = 1$ we obtain synchronization (case c) of Fig. 2).

Now we can proceed by constructing terms representing possible computations for given places of PN N. These will be bound to a corresponding E-variables in a way given by the definition:

Definition 3. *According to the structure of Petri net in the vicinity of a given place, terms are bound to corresponding variables for elementary situations depicted in figures Fig. 1 and Fig. 2 as follows:*

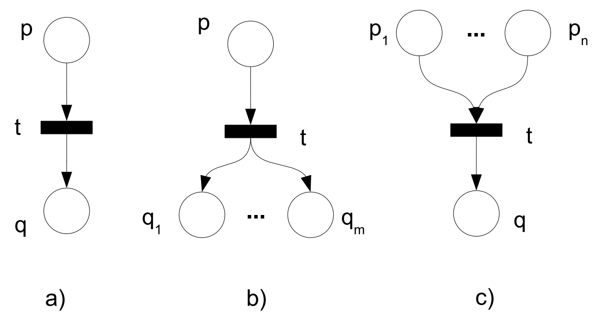

Figure 2: Basic compositions as a special cases of the general composition

- *deadlock (Fig. 1a):* $E(p) = \delta$
- *alternative composition (Fig. 1b):* $E(p) = t_1 \cdot E(q_1) + t_2 \cdot E(q_2) + ... + t_n \cdot E(q_n)$
- *sequential composition (Fig. 2a):* $E(p) = t \cdot E(q)$
- *parallel composition (Fig. 2b):* $E(p) = t \cdot (E(q_1) \parallel ... \parallel E(q_n))$
- *synchronization (Fig. 2c):* $E(p_1) = (t \cdot E(q))^{[1]}, E(p_2) = (t \cdot E(q))^{[2]}, ..., E(p_n) = (t \cdot E(q))^{[n]}$,
  and the composition function is defined:
  $\pi((t \cdot E(q))^{[1]}, ..., (t \cdot E(q))^{[n]}) = t \cdot E(q)$ *or* $\pi(E(p_1), ..., E(p_n)) = t \cdot E(q)$
- *general composition (Fig. 1c):* $E(p_1) = (t \cdot (E(q_1) \parallel ... \parallel E(q_m)))^{[1]}$,
  $E(p_2) = (t \cdot (E(q_1) \parallel ... \parallel E(q_m)))^{[2]}, ...,$
  $E(p_n) = (t \cdot (E(q_1) \parallel ... \parallel E(q_m)))^{[n]}$,
  and the composition function is defined in the following way:
  $\pi(E(p_1), ..., E(p_n)) = t \cdot (E(q_1) \parallel ... \parallel E(q_m))$
- *transition without post-place(s) (Fig. 3a):* $E(p) = t$
- *transition without pre-place(s) (Fig. 3b): a new place is added, such that firing properties of a transition given are preserved (Fig. 3c):* $E(p) = t \cdot E(q)$.

In the case of the synchronization we can observe that, all variables composed are assigned to terms representing process components instead of true processes. These components, if all are present within the term representing the net computation, can merge together by means of composition function π, and form the true process (able to execute action t and then to behave like $E(q)$).

Taking into account the case, when a transition without pre-places occurs within the net structure (Fig. 3b), the following solution is proposed: for every such transition t, a new pre-place is added, such that the firing properties of transition t are preserved (Fig. 3c). This is achieved by setting the initial marking of given place to ω, where

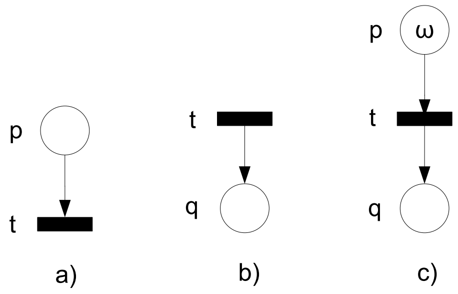

Figure 3: Transitions without input/output

$\forall n \in \mathbb{N} : \omega \pm n = \omega$. In fact, this causes the transition t can be fired infinitely many times. Combining these basic principles, we are able to construct terms for more complicated net structures.

Definition 4. *Let the PN N is given by $N = (P, T, pre, post)$, $m \in \mathbb{N}^k$ stands for its initial marking, and $k = |P|$. Then the APC semantics for N with marking m is given by the formula:*

$$\mathcal{A}(N, m) = E(p_1)^{(i_1)} \| ... \| E(p_k)^{(i_k)} \quad (4)$$

Within the Definition 4, $E(p_i)$ stands for an APC-term defined according to PN structure in the vicinity of the place p_i (Definition 3). The value i_j is given by $i_j = m(p_j)$, so it represents the marking with respect to place $p_j, 1 \le j \le k$. $E(p_j)^{(i)}$ is defined as a term $E(p_j) \| ... \| E(p_j)$, and represents a multiple (i-times) parallel composition of a process $E(p_j)$. Note that $E(p_j)^{(0)} = \delta$. When $i_j = \omega$ for place p_j, it means that $E(p_j)$ can occur infinitely many times in resulting composition.

Theorem 2. *For given PN $N = (P, T, pre, post)$, APC-term p, representing an algebraic (APC) semantics for the net N, transition $t \in T$ and m, m' markings of N, following implication holds:*

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

Proof 2. *The proof is given by the induction on a structure of the net. Let us suppose, that a step in computation of N exists: $(N, m) \xrightarrow{t} (N, m')$, then*

$$\forall p_i \in ({}^\bullet t) : m(p_i) \ge pre(p_i, t)$$

$$\forall p_i \in P : m'(p_i) = m(p_i) - pre(p_i, t) + post(p_i, t)$$

Algebraic semantics for Petri net N with marking m is given by:

$$\mathcal{A}(N, m) = E(p_1)^{(i_1)} \| ... \| E(p_k)^{(i_k)}, \ k = |P| \quad (5)$$

Transition t fired in N within a step can be of two kinds:

1. $|{}^\bullet t| = 1$
2. $|{}^\bullet t| \ge 2$

According to the transition relations of APC (Table 2), a step can be made by executing the action of the true process or by merging the process components together with execution of an action associated. Let us explore the two cases:

1. *If $|^{\bullet}t| = 1$ is the case, the situation is captured in the Fig. 4, case a). Then within the Petri net N holds: $m(p_a) \geq pre(p_a, t)$ (so the place p_a contains a token(s)) and also $m'(p_i) = m(p_i) + post(p_i, t) - pre(p_i, t), p_i \in P$ is a new marking after firing of the transition t. If $E(p_a) = t \cdot (E(p_c)\|...\|E(p_d))$ is corresponding APC semantics for process initiated in the place p_a, then a step (action t) is enabled $E(p_a)\xrightarrow{t}E(p_c)\|...\|E(p_d)$, since $E(p_a)$ is present in specification $\mathcal{A}(N, m)$. Let values $j_l, l \in \{1, ..., k\}$ are given by: $j_l = i_l - pre(p_l, t) + post(p_l, t)$. When a step $\mathcal{A}(N, m)\xrightarrow{t}\mathcal{A}(N, m')$ occurs, corresponding APC semantics of the net (N, m') is given by:*

$$\mathcal{A}(N, m') = E(p_1)^{(j_1)}\|...\|E(p_k)^{(j_k)}, \qquad (6)$$
$$k = |P|$$

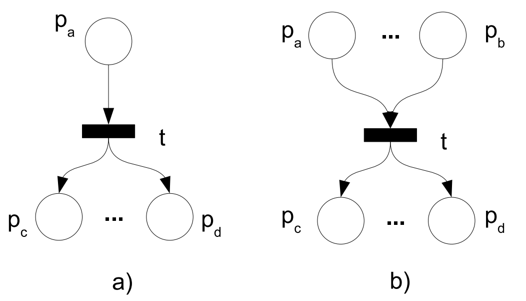

Figure 4: Two cases considered for a step in computation of N

2. *Here transition t firing in Petri net N occurs, for which $|^{\bullet}t| \geq 2$ holds. Situation is depicted in Fig. 4, case b). Within the Petri net N there holds: $m(p_a) \geq pre(p_a, t), \ldots, m(p_b) \geq pre(p_b, t)$ (so the places $p_a, \ldots, p_b$ contain enough tokens) and thus transition t can fire. From the definition of APC semantics for Petri net N (5) and Definition 3, we have that a step from (N, m) is represented by:*

$$E(p_a)^{(i_a)} \parallel ... \parallel E(p_b)^{(i_b)} \xrightarrow{t} \qquad (7)$$
$$E(p_c)^{(j_c)} \parallel ... \parallel E(p_d)^{(j_d)}$$

The step is enabled in $\mathcal{A}(N, m)$ since values $i_a, \ldots, i_b$ are given by number of tokens in corresponding places. According to the definition, variables $E(p_a)$,...,$E(p_b)$ are bound to process components and composition function is defined: $\pi(E(p_a), ..., E(p_b)) = t \cdot (E(p_c) \parallel ... \parallel E(p_d))$. The step (7) thus is enabled, and (6) holds, where: $j_1 = i_1+post(p_1, t)-pre(p_1, t)$,...,$j_k = i_k+post(p_k, t)-pre(p_k, t)$.

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

□

We give a small example here, representing the configuration of Petri net N sometimes called confusion (Fig. 5) for the sake of clarity.

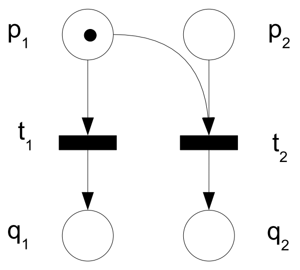

Figure 5: Confusion

First, APC-terms are assigned to variables created for every place of Petri net N.

$$E(p_1) = t_1 \cdot E(q_1) + (t_2 \cdot E(q_2))^{[1]},$$
$$E(p_2) = (t_2 \cdot E(q_2))^{[2]}$$

Next, the composition function π is defined:

$$\pi((t_2 \cdot E(q_2))^{[1]}, (t_2 \cdot E(q_2))^{[2]}) = t_2 \cdot E(q_2) \qquad (8)$$

Within the initial marking m_0 of Petri net N, the only place holding a token is p_1, so only the variable corresponding to this place will be included within the equation describing the algebraic semantics of N.

$$\mathcal{A}(N, m_0) = E(p_1) = t_1 \cdot E(q_1) + (t_2 \cdot E(q_2))^{[1]} =$$
$$t_1 \cdot E(q_1) + \delta = t_1 \cdot E(q_1)$$

In the configuration depicted, the only transition enabled is t_1 and it can fire. This is not the case of the transition t_2, because the place p_2, the pre-place of t_2 is not marked. The same could be observed within the APC representation - only one process component $((t_2 \cdot E(q_2))^{[1]})$ is present within the equation (so the mapping π cannot be used to produce a true process) and it thus cannot perform any action and is replaced by the δ.

6 An example

An example has been chosen to demonstrate a way how the transformation rules proposed can be used. The Petri net N (depicted in Fig. 6) represents a synchronization problem for sharing one resource by two processes. System modeled consists of two processes (let the first process represented by the places p_1, p_2 and transitions t_1, t_3, be named A, and the second one (p_3, p_4, t_2, t_4) be named B). The resource shared is represented by the place p_0. The token at this place indicates the resource is free to be shared either by process A or process B. The place p_2 stands for the condition 'process A is using the resource', firing transition t_1 starts the resource usage and firing t_3 ends it. Similarly, for process B, firing t_2 starts and firing t_4 ends the usage respectively. A token occurrence in the place p_3 indicates the resource is used by the process B.

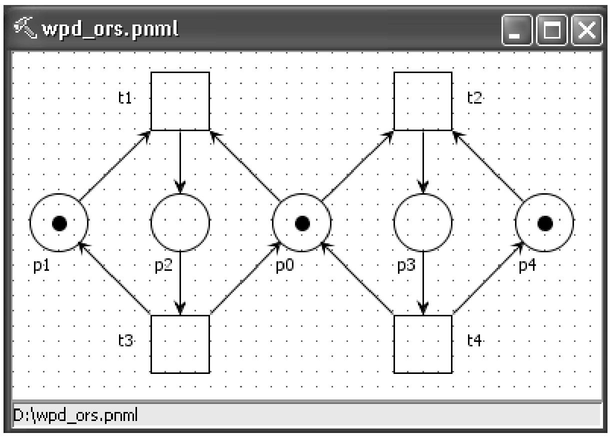

Figure 6: Petri net for resource sharing

We start with assigning APC-terms to variables created for every place of PN N, according to the structure of the net in the vicinity of the corresponding place.

$$E(p_1) = (t_1 \cdot E(p_2))^{[1]},\ E(p_2) = t_3(E(p_0) \parallel E(p_1)),\\ E(p_3) = t_4(E(p_0) \parallel E(p_4)),\ E(p_4) = (t_2 \cdot E(p_3))^{[1]},\\ E(p_0) = (t_1 \cdot E(p_2))^{[2]} + (t_2 \cdot E(p_3))^{[2]}$$

Composition function π is defined in two cases:

$$\pi((t_1 \cdot E(p_2))^{[1]}, (t_1 \cdot E(p_2))^{[2]}) = t_1 \cdot E(p_2) \qquad (9)$$

$$\pi((t_2 \cdot E(p_3))^{[1]}, (t_2 \cdot E(p_3))^{[2]}) = t_2 \cdot E(p_3) \qquad (10)$$

Since the initial marking of Petri net N is given as $m_0 = (1, 1, 0, 0, 1)$, only three places (p_0, p_1 and p_4) hold tokens, and only variables corresponding to these places will take place in equation describing the algebraic semantics of PN N.

$$\mathcal{A}(N, m_0) = E(p_0) \parallel E(p_1) \parallel E(p_4) \qquad (11)$$

Since the term on the RHS of equation (11) represents parallel composition of three processes, we expand it according to (2):

$$= E(p_0) \lfloor\!\lfloor (E(p_1) \parallel E(p_4)) + E(p_1) \lfloor\!\lfloor (E(p_0) \parallel\\ E(p_4)) + E(p_4) \lfloor\!\lfloor (E(p_0) \parallel E(p_1)) +\\ (E(p_0) ||| E(p_1)) \lfloor\!\lfloor E(p_4) + (E(p_0) ||| E(p_4)) \lfloor\!\lfloor E(p_1) +\\ (E(p_1) ||| E(p_4)) \lfloor\!\lfloor E(p_0) + (E(p_0) ||| E(p_1) ||| E(p_4))$$

After substituting terms assigned (bound) to variables $E(p_0)$, $E(p_1)$ and $E(p_4)$, we can write:

$$= [(t_1 \cdot E(p_2))^{[2]} + (t_2 \cdot E(p_3))^{[2]}] \lfloor\!\lfloor ((t_1 \cdot E(p_2))^{[1]} \parallel\\ (t_2 \cdot E(p_3))^{[1]}) +\\ (t_1 \cdot E(p_2))^{[1]} \lfloor\!\lfloor [((t_1 \cdot E(p_2))^{[2]} + (t_2 \cdot E(p_3))^{[2]}] \parallel\\ (t_2 \cdot E(p_3))^{[1]}) +\\ (t_2 \cdot E(p_3))^{[1]} \lfloor\!\lfloor [((t_1 \cdot E(p_2))^{[2]} + (t_2 \cdot E(p_3))^{[2]}] \parallel\\ (t_1 \cdot E(p_2))^{[1]}) + ([(t_1 \cdot E(p_2))^{[2]} + (t_2 \cdot E(p_3))^{[2]}] ||| (t_1 \cdot\\ E(p_2))^{[1]}) \lfloor\!\lfloor (t_2 \cdot E(p_3))^{[1]} + ([(t_1 \cdot E(p_2))^{[2]} + (t_2 \cdot\\ E(p_3))^{[2]}] ||| (t_2 \cdot E(p_3))^{[1]}) \lfloor\!\lfloor (t_1 \cdot E(p_2))^{[1]} +\\ ((t_1 \cdot E(p_2))^{[1]} ||| (t_2 \cdot E(p_3))^{[1]}) \lfloor\!\lfloor [(t_1 \cdot E(p_2))^{[2]} + (t_2 \cdot\\ E(p_3))^{[2]}] + ([(t_1 \cdot E(p_2))^{[2]} + (t_2 \cdot E(p_3))^{[2]}] ||| (t_1 \cdot\\ E(p_2))^{[1]} ||| (t_2 \cdot E(p_3))^{[1]})$$

Using the composition (π) definition (9, 10) and axioms associated (A11, A13), we have:

$$(t_1 \cdot E(p_2)) \lfloor\!\lfloor (t_2 \cdot E(p_3))^{[1]} + (t_2 \cdot E(p_3)) \lfloor\!\lfloor (t_1 \cdot E(p_2))^{[1]}$$

Using the left merge operator axiom (A9) and substituting for $E(p_2)$ and $E(p_3)$ we obtain:

$$= t_1(E(p_2) \parallel (t_2 \cdot E(p_3))^{[1]}) + t_2(E(p_3) \parallel (t_1 \cdot E(p_2))^{[1]})\\ = t_1(t_3(E(p_0) \parallel E(p_1)) \parallel\\ (t_2 \cdot E(p_3))^{[1]}) + t_2(t_4(E(p_0) \parallel E(p_4)) \parallel (t_1 \cdot E(p_2))^{[1]})$$

Considering all the cases for two processes composed by the parallel composition operator ($\parallel$) and using the axiom A6:

$$= t_1(t_3(E(p_0) \parallel E(p_1)) \lfloor\!\lfloor (t_2 \cdot E(p_3))^{[1]} +\\ (t_2 \cdot E(p_3))^{[1]} \lfloor\!\lfloor t_3(E(p_0) \parallel E(p_1)) +\\ t_3(E(p_0) \parallel E(p_1)) ||| (t_2 \cdot E(p_3))^{[1]}) +\\ t_2(t_4(E(p_0) \parallel E(p_4)) \lfloor\!\lfloor (t_1 \cdot E(p_2))^{[1]} +\\ (t_1 \cdot E(p_2))^{[1]} \lfloor\!\lfloor t_4(E(p_0) \parallel E(p_4)) +\\ t_4(E(p_0) \parallel E(p_4)) ||| (t_1 \cdot E(p_2))^{[1]})\\ = t_1(t_3(E(p_0) \parallel E(p_1)) \lfloor\!\lfloor (t_2 \cdot E(p_3))^{[1]} + \delta + \delta) +\\ t_2(t_4(E(p_0) \parallel E(p_4)) \lfloor\!\lfloor (t_1 \cdot E(p_2))^{[1]} + \delta + \delta)\\ = t_1 t_3((E(p_0) \parallel E(p_1)) \parallel (t_2 \cdot E(p_3))^{[1]}) +\\ t_2 t_4((E(p_0) \parallel E(p_4)) \parallel (t_1 \cdot E(p_2))^{[1]})$$

Since $E(p_4) = (t_2 \cdot E(p_3))^{[1]}$ and $E(p_1) = (t_1 \cdot E(p_2))^{[1]}$, we can write:

$$= t_1 t_3((E(p_0) \parallel E(p_1)) \parallel E(p_4)) +\\ t_2 t_4((E(p_0) \parallel E(p_4)) \parallel E(p_1))\\ = t_1 t_3(E(p_0) \parallel E(p_1) \parallel E(p_4)) +\\ t_2 t_4(E(p_0) \parallel E(p_1) \parallel E(p_4))$$

Using axiom A4, the term becomes even simpler:

$$= (t_1 t_3 + t_2 t_4)(E(p_0) \parallel E(p_1) \parallel E(p_4))$$

Here we can observe a parallel composition of the three variables, from which we started our derivation ($E(p_0) \parallel E(p_1) \parallel E(p_4)$). In terms of Petri nets, the initial marking was reached again. Prefix ($t_1t_3+t_2t_4$) represents the traces of processes. The sequential composition operator is often omitted, so we can state that the APC semantics is finally given by the following equation:

$$\mathcal{A}(N, m_0) = (t_1t_3 + t_2t_4)^\omega$$

7 Conclusion

In this paper a general method was presented for constructing an algebraic semantics of Petri nets, based on Algebra of Process Components (APC) by the authors. The notion of process component is introduced in order to model synchronization, which, in case of Petri nets, is modeled in a natural way. A variable is created for every place of given net and a term is bound to this variable, which express the process initiated in the corresponding place. The description of process representing computations of Petri net is given by the parallel composition of all the variables associated with places holding token(s) within the initial marking. Traces of processes can be observed in addition to changes on the PN marking along the computation. Resulting algebraic specification can further be analyzed using process algebra tools like CWB-NC, etc. A proof of identical operational behavior has also been provided.

The PETRI2APC tool, a practical implementation of method presented, is intended to be a part of multi FDT (mFDT) environment - an environment for designing and analysing of discrete systems based on three formal methods with useful complementary properties. The methods considered are Petri nets, Process algebra and B-Method. The mFDT environment is under development at DCI FEEI TU of Košice.

Acknowledgement

This work is supported in part by the VEGA 1/3140/06 and in part by the VEGA 1/4073/07 grant projects of Slovak Grant Agency.

References

[1] Baeten, J.C.M., Weijland, W.P.: *Process Algebra*, Cambridge University Press, ISBN 0 521 40043 0, pp.248, 1990.

[2] Baeten, J.C.M., Basten, T.: *Partial-Order Process Algebra*, in Handbook of Process Algebra, Elsevier Science, Amsterdam, The Netherlands, 2000, 78pp.

[3] Basten, T., Voorhoeve, M.: *An Algebraic Semantics for Hierarchical P/T Nets*, Computing Science Report, Eindhoven University of Technology, pp.32, 1995.

[4] Best, E.: *Semantics of Sequential and Parallel Programs*, Prentice Hall Europe, ISBN 0-13-460643-4, pp.352, 1996.

[5] Best, E., Devillers,R., Koutny,M.: *Petri Nets, Process Algebras and Concurrent Programming Languages* , Lecture on Petri Nets II:Applications, LNCS , Advances in Petri Nets (W.Reisig,G.Rozenberg Eds.), Springer , Berlin 1998, ISBN 3-540-65306-6,pp.1-84.

[6] Desel, J., Juhás, G., Lorenz, R.: *Process Semantics of Petri Nets over Partial Algebra*, Proceedings of ICATPN 2000, 21-st International Conference on Application and Theory of Petri Nets, vol.1825 of LNCS, pp.146-165, Springer-Verlag, 2000.

[7] Hudák, Š.: *Reachability Analysis of Systems Based on Petri Nets*, Elfa s.r.o. Košice, ISBN 80-88964-07-5, pp.272, 1999.

[8] Hudák, Š., Šimoňák, S.: *APC - Algebra of Process Components*, Proceedings of EMES'03, Oradea, Felix Spa, Romania, May 2003, pp. 57-63, ISSN 1223 - 2106.

[9] Hudák Š., Šimoňák S., Korečko Š., N.Kovacs A.: Formal Specifications and de/compositional approach to design and analysis of discrete systems, Computer Science and Technology Research Survey, Košice, Elfa, pp. 8-46, 2007.

[10] Mayr, R.: *Combining Petri nets and PA-processes*, in Theoretical Aspects of Computer Software (TACS'97), volume 1281 of Lecture Notes in Computer Science, pages 547–561, Sendai, Japan, 1997. Springer Verlag.

[11] Olderog, E.R.: *Nets, Terms and Formulas*, ISBN 0 521 40044 9, Cambridge University Press, 1991.

[12] Paige, R.F.: *Formal Method Integration via Heterogeneous Notations*, PhD Dissertation, November 1997.

[13] *PNML–Petri Net Markup Language*, URL: http://www.informatik.hu-berlin.de/ top/pnml/

[14] *PNML Framework*, URL: http://www-src.lip6.fr/logiciels/mars/PNML/

[15] 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, ISSN 1223-2106.

[16] Šimoňák, S.: *Formal methods integration based on Petri nets and process algebra transformations*, PhD Dissertation, DCI FEEI TU Košice, 2003. (In Slovak)

[17] Starke, P.H.: *Processes in Petri Nets*, LNCS 117, Fundamentals of Computation Theory, ISBN 3-540-10854-8, Springer-Verlag, 1981.

[18] *Petri Nets World (A Classification of PN)*, URL: http://www.informatik.uni-hamburg.de/TGI/PetriNets/classification/