Towards the mCRL2 Semantics for P/T Nets

Slavomír Šimoňák

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia
e-mail: slavomir.simonak@tuke.sk

***Abstract**: The aim of this paper is to present an approach for transforming Petri nets into corresponding algebraic specifications. Some limitations were discovered in our previously published method, so it was decided to employ a new approach herein. The approach, based on processes with data, transforms P/T nets into specifications in the mCRL2 process language. To underline the motivation, a brief discussion of the limitations from the previous approach is provided. Further within the paper, we define the mCRL2 algebraic semantics for P/T nets and provide a sketch of the proof. The proposed ideas were implemented in a software tool, to further enhance the potential for practical applications. Illustrative examples are provided to demonstrate the applied utilization of the proposed approach.*

Keywords: Petri nets; process algebra; formal methods; semantics; transformation

1 Introduction

Formal methods are mathematically-based techniques for a systematic specification, development, and analysis of systems. When formal methods are applied to the design and analysis of complex systems, utilization of multiple methods and corresponding verification techniques can be very useful. The reasons may include the suitability of particular formalism for the design of a given component of the system, or designer's interest in exploring specific system properties, or to manage the complexity of the considered system [1]. The importance of the formal methods integration is also underlined by the existence of successful conference series on integrated formal methods (iFM). The series is devoted to research of hybrid approaches to formal modeling and analysis based on combination of methods for system development [2].

The work presented within this paper is focused on combining Petri nets and process algebra. There has been many attempts to combine Petri nets and process algebra in the past, aiming for better exploitation of their particular advantages in modeling and analysis of systems [6][18]. Petri nets are widely adopted formal method for modeling and analysis of many types of systems. Their intuitive graphical representation can be very useful in the process of modeling and many available

techniques for examination of structural and behavioral properties are highly appreciated in the process of analysis [11]. Process algebra [3] is a symbolic formalism. Process algebraic specifications usually have no explicit representation of states and are more targeted on description of dynamic behavior of systems. Techniques of analysis available here are especially competent at comparing behavioral descriptions of concurrent systems. Such comparison can be done e.g. in the process of verification, whether the implementation of the system satisfies its (higher-level) specification [7]. In addition to the distinctions given above, the de/composition of specifications in the case of process algebra is much more natural as it is in the case of Petri nets. Therefore, Petri nets and process algebra can be considered complementary in several aspects, so there is a great potential for their combination. Formal methods integration approaches described in this paper are based on the transformations of Petri nets to corresponding algebraic specifications.

Within this paper two such approaches are described. First, is intended as an extension of our previously published approach [24], transforming ordinary Petri nets to corresponding PSF (Process Specification Formalism) specifications. In order to support some useful modeling capabilities of P/T nets, we defined some extensions and implemented them in our software tool. But within a testing stage we found a possibility of inconsistency in the behavior of the transformed system in some specific scenarios. Since we had no simple solution to this phenomenon, this line of research has been postponed and we concentrated more on the new approach. The second of the two approaches is based on processes with data and transforms (P/T) Petri nets to more recent mCRL2 language.

2 Related Work and Motivation

An active research has been performed in the field of combining Petri nets and process algebra in the recent years. In [18] a CCS-like calculus named Multi-CCS is presented. The calculus is equipped with a labeled transition system semantics and a P/T Petri net semantics using a novel approach. A class of Multi-CCS processes (well-formed finite-net Multi-CCS processes) are able to represent all finite, statically reduced P/T Petri nets [18]. A framework is presented in [6], where a net encoding can be constructed uniformly for calculi with different communication patterns. The encoding preserves several behavioral semantics. In [29] a novel compositional algebra of Petri nets is introduced. More technically, it is a compositional extension of Condition/Event nets with consume/produce loops. A net is associated with interfaces to which its transitions may be connected. Composition of such nets is performed by synchronization of transitions. In [4] authors illustrate the development of a partial-order process algebra in the style of ACP. They present an extension of given algebraic theory with a causality mechanism inspired by Petri-net theory. An approach for transforming Petri nets to a corresponding provable sequents of linear logic is presented in [23]. The authors

focus on modeling synchronization problems of mutual exclusion and deadlock within the paper. In addition to the above mentioned, a selection of notable works on the given topic would include [10][14][20][22].

Several years ago we also developed another transformation [25], which transforms a Petri net into an original APC (Algebra of Process Components) specification. When compared to our PSF-based transformation [24] it often produces simpler algebraic specifications, since APC provides special constructs for modeling the synchronization of processes. However, there is currently no tool support for analyzing APC specifications, which limits its practical applications. Our approaches working in the opposite direction (transforming algebraic specifications to Petri nets) are summarized in [27]. Practical applications of the approach may include verification of the communication protocols [25]. Properties of the proposed transformation approaches are practically examined in [28].

Comparing to the state published in [24], we updated our approach transforming (ordinary) Petri nets to corresponding algebraic specifications in several ways. Firstly, we intended to extend the approach presented in [24] (transforming Petri nets to PSF specifications) by incorporating some additional useful modeling features of P/T nets, like weights of arcs or place capacities. We incorporated some features and implemented them in a software tool (*petri2acp*), but within an extensive testing we found a possibility of inconsistency in the transformation. Regardless that fact we provide the basic ideas of this extension in the section 5 of the paper. Based on experiences with extending original transformation, we decided to employ a new approach, using processes with data. The approach (suggested also by J. F. Groote) is simpler compared to the previous one and also provides more space for further extensions. So, the main sources of motivation for this work can be formulated as follows:

- *Certain inconsistency found in our previous transformation*. Based on our practical experiments using the PSF Toolkit we have found, that there may be certain deviation in behavior described by the original Petri net and the corresponding PSF specification. The issue can appear, if multiple synchronizing transitions are present in a Petri net. And since we are interested in preserving the behavior of the original specification, we decided to address this issue.
- *New, practical, solution still needed*. Stimulating discussions with J. F. Groote, one of co-authors of the mCRL2 Toolset [12], also contributed to our belief, there is still a need for new practical solutions in this field.
- *Move to new algebraic language and more powerful toolkit*. The output format of our previous transformation [24] was PSF [15]. PSF files can be further analyzed using the PSF Toolkit, which is still available. We decided to use mCRL2 as the output format in order to use more powerful mCRL2 Toolset [13]. Both specification languages (PSF and mCRL2) are based on the same process algebra – ACP (Algebra of Communicating Processes).

3 P/T Nets

In this work, we consider P/T Petri nets according to following definition (with weights of arcs, but without place capacities), based on [21], [19]. A P/T Petri net is a 5-tuple $PN = (P, T, F, W, m_0)$, where:

$P = \{p_1, p_2, ..., p_k\}$ is a finite set of places
$T = \{t_1, t_2, ..., t_n\}$ is a finite set of transitions ($P \cap T = \emptyset$ and $P \cup T \neq \emptyset$)
$F \subseteq \{P \times T\} \cup \{T \times P\}$ is a set of arcs
$W: F \rightarrow \{1, 2, 3, ...\}$ is a weight function
$m_0: P \rightarrow \{0, 1, 2, 3, ...\}$ is the initial marking

A Petri net without initial marking is given by $N = \{P, T, F, W\}$. A Petri net with the specified initial marking m_0 is given by $N_0 = (N, m_0)$ or $N_0 = \{P, T, F, W, m_0\}$. In this work, if not given other way, we mean P/T net also by the term Petri net.

4 Process Algebra ACP

Process algebra is an algebraic framework for studying the behavior of parallel or distributed systems [3]. ACP [8] is one of the most widely-known process algebras, together with CCS and CSP. ACP emphasizes the algebraic aspect more than other two, it is an equational theory, which can be equipped with a number of semantical models. Moreover, the ACP provides a more general scheme of communication compared to CCS and CSP [3]. In this work we will use extended version of process algebra ACP, the Algebra of Communicating Processes with abstraction (ACP_τ) [9].

4.1 Syntax

The signature of ACP contains a set of constant symbols A (atomic actions), partial communication function γ on A, a special constant δ (deadlock, $\delta \notin A$) and several operators: + (alternative composition), · (sequential composition), $\|$ (parallel composition), $\lfloor\!\lfloor$ (left merge), | (communication merge), and ∂_H (encapsulation). The axiom system of process algebra ACP in style of [8] is given in Table 1. Within the table a, b, c denote atomic actions of a finite alphabet, while x, y and z are processes. Axioms A1 – A7 represent laws for processes composed using basic operators for alternative and sequential composition. Basic properties of communication are expressed by axioms C1 – C3 and further specified by the axioms CM1 – CM9. The axioms D1 – D4 are connected with the encapsulation operator ∂_H, which function is to encapsulate a process with respect to actions in the set H.

Further, we will also consider several axioms in addition to those in Table 1. The axioms in Table 2 provide the mechanism for hiding internal actions. The constant τ (silent action) is introduced, so $A_\tau = A \cup \{\tau\}$. The operation τ_I renames actions in I into τ. Here H and I are both subsets of A_τ, such that $H \subseteq A$ and $I \subseteq A - \{\delta\}$.

Table 1
Axioms of process algebra ACP [8]

$x + y = y + x$	**A1**	$a\,x \between y = a(x \parallel y)$	**CM3**
$(x + y) + z = x + (y + z)$	**A2**	$(x + y) \between z = x \between z + y \between z$	**CM4**
$x + x = x$	**A3**	$a\,x \mid b = (a \mid b) \cdot x$	**CM5**
$(x + y) \cdot z = xz + yz$	**A4**	$a \mid bx = (a \mid b) \cdot x$	**CM6**
$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	**A5**	$ax \mid by = (a \mid b) \cdot (x \parallel y)$	**CM7**
$x + \delta = x$	**A6**	$(x + y) \mid z = x \mid z + y \mid z$	**CM8**
$\delta \cdot x = \delta$	**A7**	$x \mid (y + z) = x \mid y + x \mid z$	**CM9**
$a \mid b = b \mid a$	**C1**	$\partial_H(a) = a$ if $a \notin H$	**D1**
$(a \mid b) \mid c = a \mid (b \mid c)$	**C2**	$\partial_H(a) = \delta$ if $a \in H$	**D2**
$\delta \mid a = \delta$	**C3**	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$	**D3**
$x \parallel y = x \between y + y \between x + x \mid y$	**CM1**	$\partial_H(x \cdot y) = \partial_H(x) \cdot \partial_H(y)$	**D4**
$a \between x = ax$	**CM2**		

Process algebra with axioms from both Table 1 and Table 2 is named ACP_τ in [9]. For the sake of simplicity, we use the name ACP also for ACP_τ within this work.

Table 2
Additional axioms of process algebra ACP_τ

$x\tau = x$	**T1**	$x \mid (\tau y) = x \mid y$	**TC4**
$\tau x + x = \tau x$	**T2**	$\partial_H(\tau) = \tau$	**DT**
$a(\tau x + y) = a(\tau x + y) + ax$	**T3**	$\tau_I(\tau) = \tau$	**TI1**
$\tau \between x = \tau x$	**TM1**	$\tau_I(a) = a$ if $a \notin I$	**TI2**
$(\tau x) \between y = \tau(x \parallel y)$	**TM2**	$\tau_I(a) = \tau$ if $a \in I$	**TI3**
$\tau \mid x = \delta$	**TC1**	$\tau_I(x + y) = \tau_I(x) + \tau_I(y)$	**TI4**
$x \mid \tau = \delta$	**TC2**	$\tau_I(x \cdot y) = \tau_I(x) \cdot \tau_I(y)$	**TI5**
$(\tau x) \mid y = x \mid y$	**TC3**		

4.2 Semantics

The constants in set A (atomic actions) are considered indivisible actions. The process $x \cdot y$ executes process x first and after it is finished, it starts executing y. The process $x + y$ executes either process x or process y. For the parallel composition ($x \parallel y$) within a process algebra without communication, the actions of x and y are

arbitrarily interleaved. It is assumed, that the atomic actions have no duration in time, and that two actions cannot happen simultaneously [5]. In order to specify the merge by finite number of equations, the auxiliary operator ⌊⌊ was introduced. The operator has the same meaning as the operator || with the restriction, that the first step to execute must be from the left process (e.g. from x in the case of $x \lfloor\lfloor y$). In ACP, however, for the merge of two processes $x \,||\, y$ there are three possibilities to proceed in general (CM1 in Table 1). The process either starts with a first step of x, or a first step of y or with a communication between the processes. The communication merge ($x \mid y$) is the merger of two separate processes, where the first step is a communication between the processes. The merge, is an extension of the communication function, on atomic actions ($\gamma: A \times A \rightarrow A$).

When the communication function is not defined, the communication merge equals to δ. In the case, we want to express that some actions cannot happen and should be blocked, we can rename them to δ by applying the encapsulation operator ∂_H. A process $\partial_H(x)$ can execute all actions of the process x, except the actions, which names are in the set H. An important tool in the analysis of systems is abstraction, as the names of internal events can be hidden, so the relationship between externally visible events becomes clearer. The hidden action (τ) cannot be observed directly, nor communicate with other actions [17]. In order to assign an operational semantics to process expressions, we need to determine, which actions a process can perform. The fact, that the process t can execute the action a and turn to the process s can be denoted by: $t \xrightarrow{a} s$. The symbol √ is used for representing successful termination, so expression $t \xrightarrow{a} \surd$ denotes the fact that t can terminate by executing the action a. Definitions of action relations for process algebra ACP can be found in [5].

4.3 PSF Language

PSF is a formal description technique for specification of concurrent systems [16]. PSF has been designed for a set of tools to support process algebra ACP. Process expressions in PSF can be constructed using atomic actions and operators. Such expressions in combination with recursive process definitions can be used to define processes. Basic operators on processes include sequential, alternative and parallel composition. Communication between parallel processes can be defined using the communication function. The *encapsulation* operator can be used for renaming actions to the constant indicating a deadlock (`delta`). The *hiding* operator can be used for renaming actions to the constant indicating an internal action (`skip`).

4.4 mCRL2 Language and mCRL2 Toolkit

The mCRL2 language and toolset have been developed in order to support the analysis of complex distributed systems. The mCRL2 language consists from three parts: a data language, a process language, and a property language [13]. Data and

transformations on them are described using abstract data types, which allows users to create their own data types. There is a built-in support for several data types, like booleans, natural numbers, integers and reals. More complex types can be created using type constructors like sets, lists, and functions over any data type. The behavior of a system is described by processes. Processes are composed from a set of actions and a set of operators on actions and processes. The operators include sequential, alternative and parallel composition, multi-action composition and abstraction. Primitives for modeling real-time systems are also available. Actions can be parametrized by data and conditional process behavior is supported as well. The semantics of processes is given using a structural operational semantics, which associates a labelled transition system to every expression. High-level properties can be expressed using an extension of propositional modal μ-calculus. Least and greatest fixpoint operators can be used together with modal operators for describing requirements. The mCRL2 toolset consists of several tools for analysis of systems specified using the mCRL2 language. The collection of tools is open source and is available on most popular operating systems. More details can be found in [12].

5 Extending the PSF Semantics for P/T Nets

In this section we briefly present the ideas of extending our PSF semantics for Petri nets. In order to support modeling features of P/T Nets (e.g. arc weights), we made several generalizations to the original definition of the semantics [24]. At the beginning a special variable (*E*-variable) for every place of a Petri net is defined (e.g. variable *E(p)* for the place *p*). A variable is assigned an algebraic term expressing all computations starting in the corresponding place. The particular form of the term depends on the structure of the Petri net around the place.

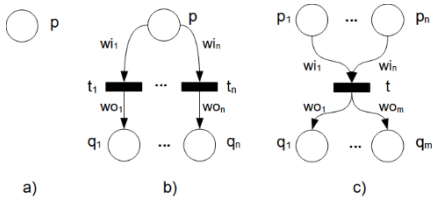


Figure 1
Petri nets - basic configurations

Basic configurations (Figure 1) are extended by considering the weights of arcs. In the case a) no arcs are connected to the place *p*, so no computations are initiated in the place and the term δ (deadlock) is assigned to the variable *E(p)*. In the case b), if a sufficient number of tokens is ready in the place *p*, a choice is made and one of the transitions $t_1, ..., t_n$ can fire. In the case c) (general composition), enough tokens must be present in all pre-places ($p_1, ..., p_n$) of the transition *t* to enable it for firing.

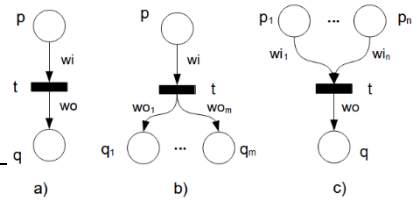


Figure 2
Three basic compositions as a special cases of the general composition

The general composition can be perceived as a generalization of three basic types of composition [24]: sequential (if $n=1$ and $m=1$), parallel (if $n=1$ and $m>1$) and the synchronization (if $n>1$ and $m=1$, Figure 2). The alternative composition, as another one of the basic types, was considered above. Now we can proceed to construction of process terms representing the possible computations in particular situations.

Definition 1. According to the structure of a Petri net in the neighborhood of the considered place, the terms for elementary configurations are defined as follows:

1. Deadlock (Figure 1 – a)): $E(p) = \delta$
2. Generalized sequential composition (Figure 2 – a)):

$$E(p) = t_{epp}, wst = \left(\underbrace{t_{eps} \parallel \ldots \parallel t_{eps}}_{wi}\right) \cdot t \cdot \left(\underbrace{E(q) \parallel \ldots \parallel E(q)}_{wo} \parallel (wst)\right),$$
$$\gamma(t_{eps}, t_{epp}) = t_{epc}, I = \{t_{epc}\}, H = \{t_{eps}, t_{epp}\}$$

3. Generalized alternative composition (Figure 1 – b)):

$$E(p) = t_{1epp} + \ldots + t_{nepp},$$
$$wst_1 = \left(\underbrace{t_{1eps} \parallel \ldots \parallel t_{1eps}}_{wi_1}\right) \cdot t_1 \cdot \left(\underbrace{E(q1) \parallel \ldots \parallel E(q1)}_{wo_1} \parallel (wst_1 + \ldots + wst_n)\right),$$
$$\ldots$$
$$wst_n = \left(\underbrace{t_{neps} \parallel \ldots \parallel t_{neps}}_{wi_n}\right) \cdot t_n \cdot \left(\underbrace{E(qn) \parallel \ldots \parallel E(qn)}_{wo_n} \parallel (wst_1 + \ldots + wst_n)\right),$$
$$\gamma(t_{1eps}, t_{1epp}) = t_{1epc}, \ldots, \gamma(t_{neps}, t_{nepp}) = t_{nepc},$$
$$I = \{t_{1epc}, \ldots, t_{nepc}\}, H = \{t_{1eps}, t_{1epp}, \ldots, t_{neps}, t_{nepp}\}$$

4. Generalized parallel composition (Figure 2 – b)):

$$E(p) = t_{epp}, \gamma(t_{eps}, t_{epp}) = t_{epc}, I = \{t_{epc}\}, H = \{t_{eps}, t_{epp}\}$$
$$wst = \left(\underbrace{t_{eps} \parallel \ldots \parallel t_{eps}}_{wi}\right) \cdot t \cdot \left(\underbrace{E(q1) \parallel \ldots \parallel E(q1)}_{wo_1} \parallel \ldots \parallel \underbrace{E(qm) \parallel \ldots \parallel E(qm)}_{wo_m} \parallel (wst)\right)$$

5. Generalized synchronization (Figure 2 – c)):

$$E(p_1) = t_{ep1p}, \ldots, E(p_n) = t_{epnp},$$

$$wst = \left(\underbrace{t_{ep1s} \parallel \ldots \parallel t_{ep1s}}_{wi_1} \parallel \ldots \parallel \underbrace{t_{epns} \parallel \ldots \parallel t_{epns}}_{wi_n} \right) \cdot t \cdot \left(\underbrace{E(q) \parallel \ldots \parallel E(q)}_{wo} \parallel (wst) \right),$$

$$\gamma(t_{ep1s}, t_{ep1p}) = t_{ep1c}, \ldots, \gamma(t_{epns}, t_{epnp}) = t_{epnc},$$

$$I = \{t_{ep1c}, \ldots, t_{epnc}\}, H = \{t_{ep1s}, t_{ep1p}, \ldots, t_{epns}, t_{epnp}\}$$

6. Composition with transition without post-places (Figure 3 – a)):

$$E(p) = t_{epp}, wst = \left(\underbrace{t_{eps} \parallel \ldots \parallel t_{eps}}_{wi} \right) \cdot t \cdot (wst),$$

$$\gamma(t_{eps}, t_{epp}) = t_{epc}, I = \{t_{epc}\}, H = \{t_{eps}, t_{epp}\}$$

7. Composition with transition without pre-places (Figure 3 – b)):

$$E(p) = t \cdot \left(E(p) \parallel \underbrace{E(q) \parallel \ldots \parallel E(q)}_{wo} \right)$$

A concept of synchronizing processes has been used in [24] in order to define the PSF semantics for synchronization composition. In this work we use generalized compositions, considering also weights of arcs, so the concept is utilized more often. This way, we can express in PSF language a P/T Net condition that a transition can only fire, when its pre-place holds enough tokens, which is given by the weight of the arc connecting them together. For example in the case of generalized sequential composition (Figure 2 - a)), a new process *wst* is defined in a way:

$$wst = \left(\underbrace{t_{eps} \parallel \ldots \parallel t_{eps}}_{wi} \right) \cdot t \cdot \left(\underbrace{E(q) \parallel \ldots \parallel E(q)}_{wo} \parallel (wst) \right)$$

This ensures, that the synchronization will precede the execution of the action *t*. To achieve the required behavior, $E(p) = t_{epp}$ and the communication function is defined: $\gamma(t_{eps}, t_{epp}) = t_{epc}$. So the actions t_{eps} and t_{epp} can communicate together and the result will be action t_{epc}. The action is further hidden (renamed to τ, by including it to the set of internal actions *I*) to make the synchronizing actions invisible. A set of actions to be encapsulated (renamed to δ) is defined to force the communication of t_{eps} and t_{epp} actions, instead of their individual execution: $H = \{t_{eps}, t_{epp}\}$.

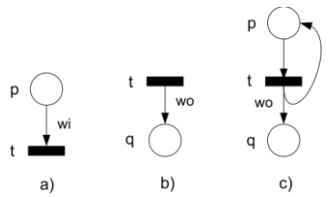


Figure 3
Transitions without input/output places

In the case, when a transition without pre-places is present within the Petri net (Figure 3 - b)), we need to preserve the firing properties of such transition. Since the transition can fire infinitely, we propose the solution, where for every such transition a new pre-place is created and connected to the transition according to the Figure 3 - c). By combining the basic principles introduced so far, we are able to construct the corresponding terms for more complicated net structures.

5.1 Enhanced PSF Semantics for P/T Nets

Within this subsection we are going to define PSF semantics for P/T nets.

Definition 2. Let the P/T net be given by the $N = (P, T, F, W)$. The algebraic PSF semantics for the P/T net N and the marking m is given by:

$$A(N,m) = \tau_I\left(\partial_H\left((wst_1+\ldots+wst_s) \between E(p_1)^{(i_1)} \between \ldots \between E(p_k)^{(i_k)}\right)\right) \quad (1)$$

Considering the equation 1, $E(p_j)$ represents a process term, constructed according to the Definition 1. The value i_j, given by $i_j = m(p_j)$ represents the initial marking in the place p_j, $1 \leq j \leq k$. By the $E(p_j)^{(i)}$ we mean the term $E(p_j)||\ldots||E(p_j)$, representing a multiple (i-times) parallel composition of a process $E(p_j)$ itself. Synchronizing processes wst_i, $1 \leq i \leq s$, where s is the total number of synchronizing processes in the system are composed to the rest of the system. We do not provide the proof of correctness of the PSF semantics defined above. It is given here mainly as the motivation for defining the new (mCRL2-based) semantics later within this work.

5.2 PSF Semantics for P/T Nets - Incorrect Behavior

Incorrect behavior of the generated PSF specification was not easy to spot. It was only possible to observe in the special situation with multiple synchronizing transitions. To illustrate this kind of behavior, let us consider a simple Petri net as depicted in the Figure 4. There are three synchronizing transitions in the net (t_0, t_1, t_2) and two places, which are pre-places to more than one transition (p_3 and p_7).

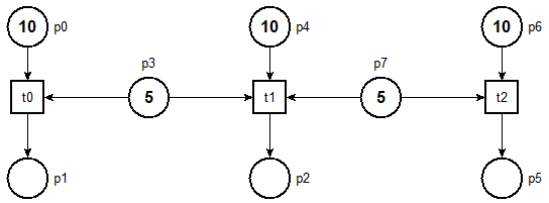
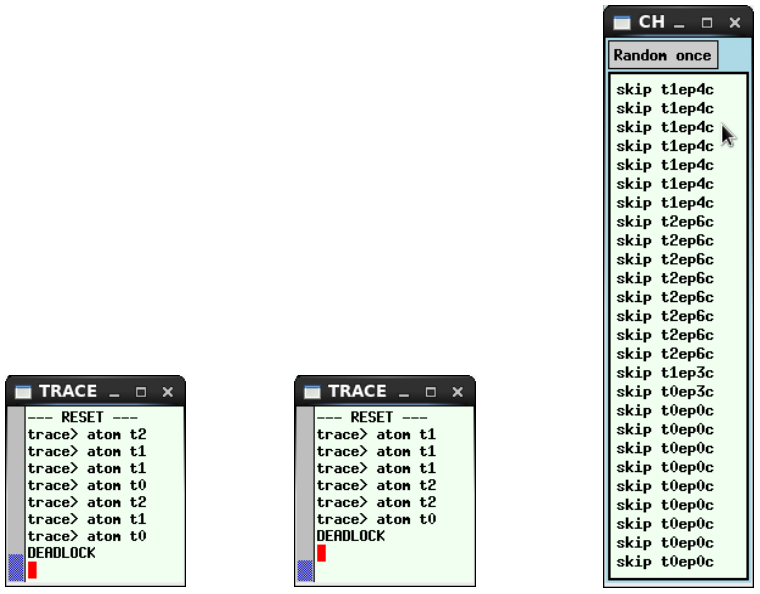


Figure 4
Sample Petri net with multiple synchronizing transitions

Considering the analysis of given Petri net, places p_1, p_2 and p_5 can be considered as counters of firing transitions t_0, t_1 and t_2 respectively. And since there is "enough" tokens in the places p_0, p_4 and p_6, we can concentrate on the places p_3 and p_7. Firing

the transition t_0 takes one token form p_3, similarly firing the transition t_2 takes a token from p_7. However, firing the transition t_1 consumes one token from both p_3 and p_7. No transition is enabled for firing (deadlock), when both p_3 and p_7 have no tokens. So if $m(p_3) = 0$ and $m(p_7) = 0$, then we can (using P-invariant equations generated e.g. by the PIPE tool) derive the following properties: $m(p_1) = m(p_5)$ and $m(p_2) = 5 - m(p_1)$. The corresponding PSF specification is given in listing below. It was generated using the updated transformation, as it is described in section 5.

```
process module SYSTINAaltsync04P2A070
begin
  atoms
    t2ep7p,t1ep7c,t2ep6p,t0ep0p,t0ep3s,t0ep3p,t1ep4c,t2ep6s,t2ep7s,t1ep3c,t0ep0s,
    t1ep4p,t1ep7s,t1ep3p,t0ep3c,t1ep4s,t2ep6c,t1ep7p,t2ep7c,t0,t1ep3s,t0ep0c,t1,t2
  processes
    ep0,ep2,ep1,ep4,ep3,ep6,ep5,ep7,wst2,wst1,wst0,Sys
  sets of atoms
    H = {t2ep7p,t2ep6p,t0ep0p,t0ep3s,t0ep3p,t2ep6s,t2ep7s,t0ep0s,t1ep4p,
    t1ep7s,t1ep3p,t1ep4s,t1ep7p,t1ep3s}
    I = {t1ep7c,t0ep3c,t1ep4c,t2ep6c,t2ep7c,t1ep3c,t0ep0c}
  communications
    t0ep3s|t0ep3p = t0ep3c
    t1ep3s|t1ep3p = t1ep3c
    t0ep0s|t0ep0p = t0ep0c
    t2ep7s|t2ep7p = t2ep7c
    t1ep4s|t1ep4p = t1ep4c
    t1ep7s|t1ep7p = t1ep7c
    t2ep6s|t2ep6p = t2ep6c
  definitions
    ep5 = delta
    ep2 = delta
    ep1 = delta
    ep4 = t1ep4p
    ep6 = t2ep6p
    ep3 = t1ep3p + t0ep3p
    ep0 = t0ep0p
    ep7 = t1ep7p + t2ep7p
    wst1 = (t1ep7s||t1ep4s||t1ep3s) . t1 . (ep2||(wst1+wst2+wst0))
    wst2 = (t2ep7s||t2ep6s) . t2 . (ep5||(wst1+wst2+wst0))
    wst0 = (t0ep0s||t0ep3s) . t0 . (ep1||(wst1+wst2+wst0))
    Sys = hide(I,encaps(H,((wst0+wst1+wst2)||ep4||ep4||ep4||ep4||ep4||ep4
    ||ep4||ep4||ep4||ep4||ep6||ep6||ep6||ep6||ep6||ep6||ep6||ep6||ep6
    ||ep6||ep3||ep3||ep3||ep3||ep3||ep0||ep0||ep0||ep0||ep0||ep0||ep0
    ||ep0||ep0||ep0||ep7||ep7||ep7||ep7||ep7)))
end SYSTINAaltsync04P2A070
```



a) b) c)

Figure 5
PSF Toolkit – simulation

Simulations of the PSF specification were performed using the PSF Toolkit. Except a number of valid runs, like the one in Figure 5, case a), we also observed, time to time, some which were not fully consistent with expected behavior. Example of this behavior is a premature deadlock (Figure 5, case b)). We observed this kind of behavior when we selected "incorrect" internal action, i.e. the one that would lead to execute the action which is not enabled for execution (as is the internal action *t1ep4c* in Figure 5, case c) which would lead to situation like in Figure 5, case b)).

6 mCRL2 Algebraic Semantics for P/T Nets

In this section a description of the proposed transformation is provided. The main idea of the transformation is to define a model (in a form of process specification), which would be able to simulate the behavior of the original Petri net. In the model, we need a notion of state of the Petri net given by its marking. We also define actions (corresponding to the Petri net transitions) with the ability to modify that state according to transition firing rules. So the important part is the definition of guards specifying conditions, when certain action can be executed. And, after the execution of particular action, we need to update the current state in the model appropriately. We also defined special data structures (structured sorts) for representing entities

like marking and transition vectors. To represent a marking of a Petri net with four places (`p1`, `p2`, `p3`, `p4`), we can use the following mCRL2 definition:

```
sort MTup = struct mtuple(p1:Nat, p2:Nat, p3:Nat, p4:Nat);
```

Similarly, transition vectors can be represented as follows:

```
sort TTup = struct ttuple(mp1:Int, mp2:Int, mp3:Int, mp4:Int);
```

Except these definitions we would need to define additional three sections of mCRL2 specification file: `act`, `proc` and `init`. Section `act` provides the names of actions appearing in the process definition. Section `proc` contains definitions of processes. In our case only one process will be used, for simulating the behavior of the considered Petri net. It will be a recursive process with several data parameters.

Let the process name will be `Sys` with parameters including current marking of a Petri net, transition vectors and transition in-vectors. Transition vectors are required for updating the marking after firing of particular transition. Transition vectors alone, however, would not be enough if we want to model the behavior of Petri nets with self-loops properly. And since we intend to support also this kind of Petri nets, we provide an additional set of transition vectors as parameters (transition in-vectors) as well. Transition in-vector of a particular transition represents weights of arcs incoming to the transition. So, for example, if we consider a Petri net with three transitions (`t1`, `t2`, `t3`), then the `Sys` process parameters would be the following:

```
Sys(M:MTup, Tt1,Tt2,Tt3,TIt1,TIt2,TIt3:TTup)
```

In the above declaration, *M* is the actual marking, `Tt1`, `Tt2`, `Tt3` are transition vectors and `TIt1`, `TIt2`, `TIt3` are in-vectors of the three transitions. Further we use guards expressing conditions for enabling transitions to fire. The conditions are based on actual marking and non-zero elements of in-vector of particular transition, e.g.:

```
(p1(M) >= abs(mp1(TIt1)) && p2(M) >= abs(mp2(TIt1)))
```

Here, `mp1(TIt1)` and `mp2(TIt1)` represent first two elements of in-vector of the transition `t1`. When the condition of the guard is met, action corresponding to given transition can be executed and actual state (marking) needs to be updated.

```
t1.Sys(mtuple(Int2Nat(p1(M)+mp1(Tt1)),Int2Nat(p2(M)+mp2(Tt1)),Int2Nat
(p3(M)+mp3(Tt1)),Int2Nat(p4(M)+mp4(Tt1))),Tt1,Tt2,Tt3,TIt1,TIt2,TIt3)
```

So in the above example, after executing the action `t1`, execution continues again as process `Sys`, with updated parameter representing the actual marking. The `init` section provides the initial values of the initial marking, transition vectors and transition in-vectors according to the original Petri net configuration. We provide a small illustrative example (Figure 6) and the corresponding mCRL2 specification.

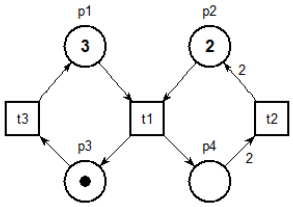


Figure 6
Simple P/T net - illustrative example

```
act
    t1;
    t2;
    t3;
proc
    Sys(M:MTup, Tt1,Tt2,Tt3,TIt1,TIt2,TIt3:TTup) =
    (p1(M) >= abs(mp1(TIt1)) && p2(M) >= abs(mp2(TIt1))) ->
    t1 . Sys(mtuple(Int2Nat(p1(M)+mp1(Tt1)), Int2Nat(p2(M)+mp2(Tt1)),
    Int2Nat(p3(M)+mp3(Tt1)), Int2Nat(p4(M)+mp4(Tt1))),
    Tt1, Tt2, Tt3, TIt1, TIt2, TIt3)
    +
    (p4(M) >= abs(mp4(TIt2))) ->
    t2 . Sys(mtuple(Int2Nat(p1(M)+mp1(Tt2)), Int2Nat(p2(M)+mp2(Tt2)),
    Int2Nat(p3(M)+mp3(Tt2)), Int2Nat(p4(M)+mp4(Tt2))),
     Tt1, Tt2, Tt3, TIt1, TIt2, TIt3)
    +
    (p3(M) >= abs(mp3(TIt3))) ->
    t3 . Sys(mtuple(Int2Nat(p1(M)+mp1(Tt3)), Int2Nat(p2(M)+mp2(Tt3)),
    Int2Nat(p3(M)+mp3(Tt3)), Int2Nat(p4(M)+mp4(Tt3))),
    Tt1, Tt2, Tt3, TIt1, TIt2, TIt3);
init
    Sys(mtuple(3,2,1,0),
    ttuple(-1,-1,1,1), ttuple(0,2,0,-2), ttuple(1,0,-1,0),
    ttuple(-1,-1,0,0), ttuple(0,0,0,-2), ttuple(0,0,-1,0));
```

Now we define the above construction more formally:

Definition 3. mCRL2 specification for the Petri net $N = (P, T, F, W, m_0)$

1. Two structured sorts are defined for representing marking (`mtuple`) and transition vectors (`ttuple`), $k = |P|$ is the number of places in the Petri net:

   `sort MTup = struct mtuple(`p_1`:Nat, `p_2`:Nat, ... , `p_k`:Nat);`

   `sort TTup = struct ttuple(`mp_1`:Int, `mp_2`:Int, ... , `mp_k`:Int);`

2. A section `act` is created in the mCRL2 specification file, with a list of action names used in the process definition. Action names correspond to the names of transitions in the Petri net, and n is the number of transitions ($n = |T|$).

   t_1; t_2; ... t_n;

3. Process `Sys` is defined in section `proc`, simulating the behavior of the Petri net:

   `Sys(M:MTup, `Tt_1`,`Tt_2`,...,`Tt_n`,`TIt_1`,`TIt_2`,...,`TIt_n`:TTup)`

   The process has several parameters. M is a vector with the actual marking of the Petri net. Tt_1, ..., Tt_n are transition vectors, representing the effect of firing of the particular transition to the marking, where t_1, ..., t_n are transition names. Transition in-vectors (TIt_1, ..., TIt_n), represent the weights of incoming arcs.

4. For each Petri net transition an alternative branch in the process `Sys` is created. The particular branch consists of two parts: the *guard*, expressing the condition enabling the branch to execute, and the corresponding *action*. After executing the action, execution continues as the process `Sys` again, with updated marking.

   (a) Guards are defined for each branch of the process, such that execution of the branch is only permitted, when the corresponding transition is enabled for firing. Condition for t_j ($1 < j < n$) is based on the actual marking M and non-zero elements of in-vector of the transition t_j.

   $$(p_{i1}(M) >= abs(mp_{i1}(TIt_j)) \;\&\&\; \ldots \;\&\&\; p_{il}(M) >= abs(mp_{il}(TIt_j)))$$

   $p_{i1}(M)$ is the value of marking in the place p_{i1}, and p_{i1} ... p_{il} are the places we need to consider in the condition (pre-places of the given transition). $mp_{i1}(TIt_j)$ is the weight of the arc leading from the place p_{i1} to the transition t_j. Since the non-zero elements of the transition in-vectors have negative values, we use the absolute value (`abs`) here.

   (b) After executing an action corresponding to the particular transition (t_j), marking is updated according to its change in the Petri net.

   $$t_j \;.\; Sys(mtuple(Int2Nat(p_1(M)+mp_1(Tt_j)), \ldots, Int2Nat(p_k(M)+mp_k(Tt_j))), Tt_1, \ldots, Tt_n, TIt_1, \ldots, TIt_n)$$

   Values of particular elements (mp_i) of the transition vector (Tt_j) are added to the corresponding elements ($p_i(M)$) of the modeled marking M, so $1 < j < n$, $1 < i < k$, and $k = |P|$. `Int2Nat` conversion is used, as the elements of the marking vector (`mtuple`) are of the `Nat` type.

5. Initialization of the `Sys` process parameters is provided within the `init` section of the specification, based on the original Petri net configuration.

   $$Sys(mtuple(m(p_1),m(p_2),\ldots,m(p_k)), ttuple(t_1v_1,t_1v_2,\ldots,t_1v_k), \ldots, ttuple(t_nv_1,t_nv_2,\ldots,t_nv_k), ttuple(t_1iv_1,t_1iv_2,\ldots,t_1iv_k), \ldots, ttuple(t_niv_1,t_niv_2,\ldots,t_niv_k));$$

   Where $m(p_i)$ denotes the initial marking in the place p_i, $1 < i < k$ and $k = |P|$. Further, t_jv_i denotes the i-th element of vector of the transition t_j, and t_jiv_i denotes the i-th element of in-vector of the transition t_j, $1 < j < n$ and $n = |T|$.

The following theorem summarizes the corresponding behavior of the original Petri net and its algebraic (mCRL2) representation.

Theorem 1. For given Petri net $N = (P, T, F, W, m_0)$, its algebraic mCRL2 representation has the same operational behavior.

Proof. Sketch of proof of the Theorem 1 is based on exploration of the possibility of mutual simulation of a Petri net N and its corresponding mCRL2 specification (given by the process `Sys`). First we will manifest the possibility of simulating steps in N by the corresponding mCRL2 specification and then, in the opposite direction, the possibility of simulating steps of mCRL2 specification by the Petri net N.

We will seek for answers to the following questions:

- If there is a transition enabled in Petri net *N*, is it also possible to execute the corresponding action (branch) of the `Sys` of the algebraic specification?
  - Initial marking of the Petri net *N* and its model in mCRL2 specification are the same (Definition 3, section 5).
  - Firing a transition in *N* and executing the corresponding action of the `Sys` have the same effect on the marking (Definition 3, section 4 (b)).
  - In the particular state the set of enabled transitions in *N* and the set of executable actions of the `Sys` are the same (Definition 3, section 4 (a)).
- If a transition is fired in Petri net *N*, does the resulting marking correspond to the one obtained by executing the matching branch of the `Sys` process?
  - In the particular state the set of enabled transitions in *N* and the set of executable actions of the `Sys` are the same (Definition 3, section 4 (a)).
  - Firing a transition in *N* and executing the corresponding action of the `Sys` have the same effect on the marking (Definition 3, section 4 (b)).

And vice versa, simulating steps of mCRL2 specification by the Petri net *N*.

- If it is possible to execute an action (branch) in the mCRL2 specification, is the corresponding transition in the Petri net *N* enabled too?
  - In the particular state the set of enabled transitions in *N* and the set of executable actions of the `Sys` are the same (Definition 3, section 4 (a)).
- If the (action) branch of the `Sys` is executed, does the resulting state correspond to the marking of *N* obtained by firing the associated transition?
  - In the particular state the set of enabled transitions in *N* and the set of executable actions of the `Sys` are the same (Definition 3, section 4 (a)).
  - Firing a transition in *N* and executing the corresponding action of the `Sys` have the same effect on the marking (Definition 3, section 4 (b)).

7 Examples

To demonstrate the practical utilization of the proposed approach, we will consider Petri nets from the Figure 6 and Figure 4, respectively. For analysis we can use the selected tools from the mCRL2 toolset. A simulation tool LpsXSim can be used for the basic insight into the system behavior. To investigate more specific properties, the model checking approach can be very useful. For the Petri net from the Figure 6, the corresponding mCRL2 specification is given in section 6. To check if the system is deadlock-free, the Mu-calculus formula `[true*]<true>true` can be used.

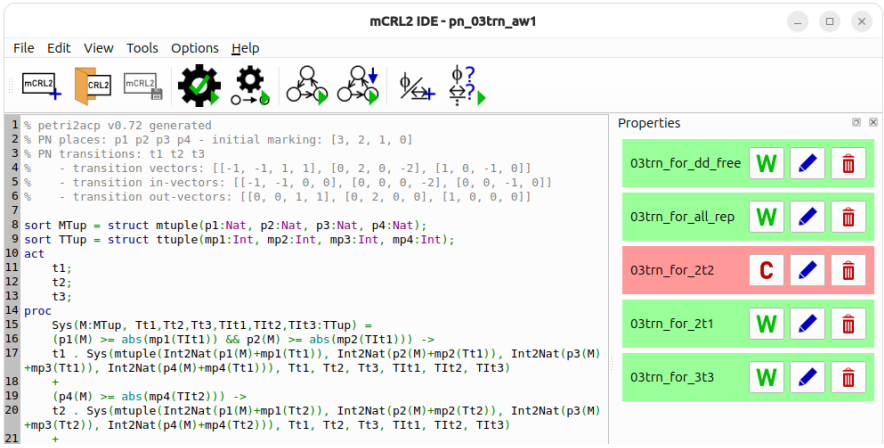


Figure 7
Verification of Mu-calculus formulas in mCRL2 IDE

An interesting question may be whether all the actions (`t1`, `t2`, `t3`) can be eventually executed within a computation starting in any state, as expressed by the formula `[true*][true*.t1.true*.t2.true*.t3.true*]true`, named *03trn_for_all_rep* in the Figure 7. To check, if the given action can be executed consecutively several times, we can use formulas like: `[true*]<true*.t1.t1.true*>true`, `[true*]<true*.t2.t2.true*>true`, or `[true*]<true*.t3.t3.t3.true*>true`. Formulas are named *03trn_for_2t1*, *03trn_for_2t2*, and *03trn_for_3t3,* respectively in the Figure 7. As we can see, all the formulas evaluate to `true`, except the *03trn_for_2t2*. This means, that the action `t2` cannot be executed twice in a row.

Using the Petri net from the Figure 4 we demonstrated an incorrect behavior of the corresponding PSF specification (Figure 5). As a kind of validation, we translated the mentioned Petri net into the mCRL2 language using the approach proposed in section 6. The resulting specification is given in the following listing (formatted).

```
% petri2acp v0.72 generated
% PN places: p0 p1 p2 p3 p4 p5 p6 p7 - initial marking: [10, 0, 0, 5, 10, 0, 10, 5]
% PN transitions: t0 t1 t2
% - transition vectors: [[-1, 1, 0, -1, 0, 0, 0, 0], [0, 0, 1, -1, -1, 0, 0, -1],
 [0, 0, 0, 0, 0, 1, -1, -1]]
% - transition in-vectors: [[-1, 0, 0, -1, 0, 0, 0, 0], [0, 0, 0, -1, -1, 0, 0, -1],
 [0, 0, 0, 0, 0, 0, -1, -1]]
% - transition out-vectors: [[0, 1, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0, 1, 0, 0]]

sort MTup = struct mtuple(p0:Nat, p1:Nat, p2:Nat, p3:Nat, p4:Nat, p5:Nat, p6:Nat,
 p7:Nat);
sort TTup = struct ttuple(mp0:Int, mp1:Int, mp2:Int, mp3:Int, mp4:Int, mp5:Int,
 mp6:Int, mp7:Int);
act
 t0;
 t1;
 t2;
proc
 Sys(M:MTup, Tt0,Tt1,Tt2,TIt0,TIt1,TIt2:TTup) =
 (p0(M) >= abs(mp0(TIt0)) && p3(M) >= abs(mp3(TIt0))) ->
```

```
 t0 . Sys(mtuple(Int2Nat(p0(M)+mp0(Tt0)), Int2Nat(p1(M)+mp1(Tt0)),
  Int2Nat(p2(M)+mp2(Tt0)), Int2Nat(p3(M)+mp3(Tt0)), Int2Nat(p4(M)+mp4(Tt0)),
  Int2Nat(p5(M)+mp5(Tt0)), Int2Nat(p6(M)+mp6(Tt0)), Int2Nat(p7(M)+mp7(Tt0))),
  Tt0, Tt1, Tt2, TIt0, TIt1, TIt2)
        +
 (p3(M) >= abs(mp3(TIt1)) && p4(M) >= abs(mp4(TIt1)) && p7(M) >= abs(mp7(TIt1))) ->
 t1 . Sys(mtuple(Int2Nat(p0(M)+mp0(Tt1)), Int2Nat(p1(M)+mp1(Tt1)),
  Int2Nat(p2(M)+mp2(Tt1)), Int2Nat(p3(M)+mp3(Tt1)), Int2Nat(p4(M)+mp4(Tt1)),
  Int2Nat(p5(M)+mp5(Tt1)), Int2Nat(p6(M)+mp6(Tt1)), Int2Nat(p7(M)+mp7(Tt1))),
  Tt0, Tt1, Tt2, TIt0, TIt1, TIt2)
        +
 (p6(M) >= abs(mp6(TIt2)) && p7(M) >= abs(mp7(TIt2))) ->
 t2 . Sys(mtuple(Int2Nat(p0(M)+mp0(Tt2)), Int2Nat(p1(M)+mp1(Tt2)),
  Int2Nat(p2(M)+mp2(Tt2)), Int2Nat(p3(M)+mp3(Tt2)), Int2Nat(p4(M)+mp4(Tt2)),
  Int2Nat(p5(M)+mp5(Tt2)), Int2Nat(p6(M)+mp6(Tt2)), Int2Nat(p7(M)+mp7(Tt2))),
  Tt0, Tt1, Tt2, TIt0, TIt1, TIt2);
init
 Sys(mtuple(10,0,0,5,10,0,10,5),ttuple(-1,1,0,-1,0,0,0,0),ttuple(0,0,1,-1,-1,0,0,-1),
  ttuple(0,0,0,0,0,1,-1,-1), ttuple(-1,0,0,-1,0,0,0,0), ttuple(0,0,0,-1,-1,0,0,-1),
  ttuple(0,0,0,0,0,0,-1,-1));
```

Using the LpsXSim tool (Figure 8) we found, there is no premature deadlock in the situation from the Figure 5, case b), as it was in the case of the PSF specification. Further we will use model checking to verify some of the system properties. By the formula `<t1.t1.t1.t2.t2.t0>true`, we can verify if there exists a path leading to a state, where `true` is valid. So we want to know if an action can be performed after the given sequence of actions. The formula evaluates to `true`. The formula `<t1.t1.t1.t1.t1.t1>true` can be used to check the possibility to perform six actions `t1` in a row. The formula evaluates to `false`. Indeed, within the system neither of the actions (`t0`, `t1`, `t2`) can be performed more than five times.

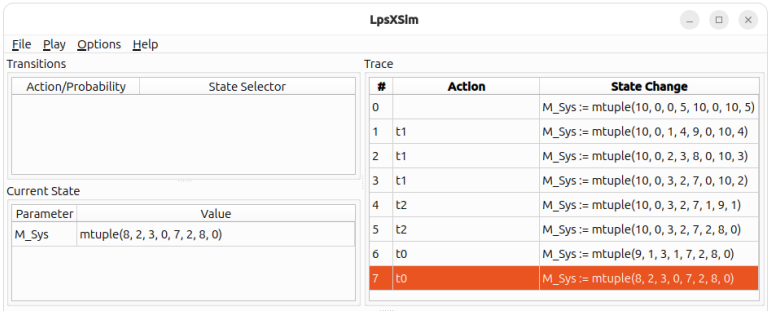


Figure 8
LpsXSim simulation

8 Conclusions

This paper proposed the mCRL2 algebraic semantics, for P/T Nets. After introducing the basic principles, it was defined more precisely and provided a sketch of the proof as well. The proposed ideas were implemented within a developed software tool that demonstrated the viability of the concept through targeted examples. Using this approach, for a system specified by the P/T net, one can obtain the corresponding representation of the system in the process language mCRL2.

This way one can (almost effortlessly) gain another view on the considered system and utilize the benefits of the new representation and the available tools.

Comparing with our previous work on PSF algebraic semantics for Petri nets, the proposed solution provides several advantages:

- Correct behavior also in the cases where the previous solution failed
- More flexible and easier extensible
- mCRL2 is very powerful process language, with the extensive toolset

Comparing with some of the more theoretical approaches mentioned in section 2, the work presented within this paper aims to be more practically oriented. Many of the available solutions are based on process algebras like CCS, or they can use their own process languages. We have chosen the mCRL2 language, which is based on the process algebra ACP. While CCS uses model-based approach, the ACP is based on equational style of reasoning. So within the ACP, the base point is an equational theory, which may have diverse semantic interpretations [7][3]:

- Widely-adopted process language, without defining specialized extensions used as a destination language ensures solid support by the existing tools.
- Available implementation in a software tool provides a potential for practical utilization of the solution.

More research is required to fully assess the properties and benefits of the proposed approach. In the future, consideration to additional extensions (e.g. support for inhibitor arcs, place capacity, processes with data, or time relations) should be addressed. Another possible direction would be the support for verification of properties involving the marking of the Petri net. In such a case, one could incorporate additional information on the marking into the actions corresponding to Petri net transitions. Expressions involving values of the marking, then could be used in μ-calculus formulas in a more straightforward manner.

Acknowledgement

I would like to express my gratitude to J.F. Groote for stimulating communications on the topic of the paper.

References

[1] Ábrahám, E., Huisman, M. (Eds.): *Integrated Formal Methods*, 12th International Conference, IFM 2016. doi.org/10.1007/978-3-319-33693-0

[2] The International Conference on integrated Formal Methods, Home page. www.ifmconference.org/

[3] Baeten, J.C.M.: A Brief History of Process Algebra. Theoretical Computer Science, Vol. 335, no. 2-3, 2005. doi.org/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 Handbook of Process Algebra, Elsevier Science, Amsterdam, 2001. doi.org/10.1016/B978-044482830-9/50031-X

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

[6] 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, LNCS 9465, 2015. doi.org/10.1007/978-3-319-25527-9_8

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

[8] Bergstra, J.A., Klop, J.W.: Process algebra for synchronous communication. Information and Control, 60(1–3), 1984. doi.org/10.1016/S0019-9958(84)80025-X

[9] Bergstra, J.A., Klop, J.W.: Algebra of communicating processes with abstraction. Theoretical Computer Science, Volume 37, 1985. doi.org/10.1016/0304-3975(85)90088-X

[10] Best, E., Devillers, R., Koutny, M.: Petri Nets, Process Algebras and Concurrent Programming Languages, Lectures on Petri Nets II: Applications, LNCS, Springer, 1998. doi.org/10.1007/3-540-65307-4_46

[11] 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, LNCS 7480, 2013. doi.org/10.1007/978-3-642-38143-0_5

[12] Bunte, O. et al.: The mCRL2 Toolset for Analysing Concurrent Systems. In: Vojnar, T., Zhang, L. (eds) Tools and Algorithms for the Construction and Analysis of Systems, 2019, Springer. doi.org/10.1007/978-3-030-17465-1_2

[13] Cranen, S. et al.: 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, LNCS 7795, 2013. doi.org/10.1007/978-3-642-36742-7_15

[14] Desel, J., Juhás, G., Lorenz, R.: Process Semantics of Petri Nets over Partial Algebra, Proceedings of ICATPN 2000, Application and Theory of Petri Nets, LNCS 1825, Springer, 2000. doi.org/10.1007/3-540-44988-4_10

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

[16] Diertens, B.: New features in PSF I - Interrupts, Disrupts, and Priorities, Rep. P9417, Programming Research Group, University of Amsterdam, 1994.

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

[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, 2015. doi.org/10.1007/978-3-319-25527-9_17

[19] Hudák, Š.: Reachability Analysis of Systems Based on Petri Nets, Elfa, 1999.

[20] Lodaya, K.: A regular viewpoint on processes and algebra, Acta Cybernetica 17, 751-763, 2006.

[21] Murata, T.: Petri nets: Properties, analysis and applications, in Proceedings of the IEEE, Vol. 77, No. 4, 1989.

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

[23] Perháč, J., Mihályi, D, Novitzká, V.: Modeling Synchronization Problems: From Composed Petri Nets to Provable Linear Sequents, Acta Polytechnica Hungarica, 14(8), 2017. doi.org/10.12700/APH.14.8.2017.8.9

[24] Šimoňák, S., Tomášek, M.: ACP Semantics for Petri Nets. Computing and Informatics, 2018, 37(6). doi.org/10.4149/cai_2018_6_1464

[25] Šimoňák, S. Hudák, Š. Korečko, Š.: APC Semantics for Petri Nets, Informatica, 32(3), 2008. https://www.informatica.si

[26] Šimoňák, S.: Verification of Communication Protocols Based on Formal Methods Integration, Acta Polytechnica Hungarica, 9(4), 2012. https://acta.uni-obuda.hu/Simonak_36.pdf

[27] Šimoňák, S. et al.: Abstraction-enriched Formal Methods Integration, Acta Polytechnica Hungarica, 15(7), 2018. doi.org/10.12700/APH.15.7.2018.7.9

[28] Šimoňák, S., Harvilík, D.: Practical Examination of Formal Methods Transformations Properties, Acta Polytechnica Hungarica, 19(5), 2022. doi.org/10.12700/APH.19.5.2022.5.3

[29] Sobocinski, P.: Representations of Petri net interactions, CONCUR 2010 - Concurrency Theory, LNCS 6269. doi.org/10.1007/978-3-642-15375-4_38