Practical Examination of Formal Methods Transformations Properties

Slavomír Šimoňák, Daniel Harvilík

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

Abstract: The paper is focused on examination of properties of transformations of Petri nets and process algebra specifications. After a brief introduction to formal methods and the transformations used, we provide descriptions of several experiments regarding Petri nets and process algebra transformations we accomplished using our transformation tools. The motivations behind this research are to practically evaluate the benefits we would gain by transformation in the field of analysis of resulting specification as well as to verify the accuracy and correctness of the tools by performing the transformations in both directions. By evaluating the experiments we were able to better perceive the actual state of the tools in practical level, the role of transformations in the field of formal methods integration and to collect some suggestions which may stimulate our further research in the given field.

Keywords: formal methods; Petri nets; process algebra; transformations; ACP; analysis

1 Introduction

Formal methods are mathematically rigorous techniques used in development of software, hardware and hybrid systems. The use of formal methods is motivated by the expectation that performing appropriate mathematical analysis can contribute to the reliability, correctness, and safeness of designed system. Formal methods can be applied in any development phase: specification, verification and implementation [1]. The basic element of the formal method is formal notation. It is a language that has formally defined syntax and semantics [2] [3]. Using formal notation a specification can be written so that their expression is clear and unambiguous, allowing defining critical aspects of the system [4]. The advantages of using formal methods are highlighted in [4], which include precision, conciseness, abstraction, and reasoning.

Combining several formal methods within development of a system can provide different views on the system and access to methods and tools of particular formalisms. At the development of complex systems it seems to be advantageous

applying different methods and approaches for modeling of their particular aspects and thus ensuring optimal conditions for development of such systems [5] [6]. From possible integration approaches [7] [8] we focus on the transformations between selected formalisms within this work. In the following section we briefly introduce two formalisms (Petri nets and process algebra ACP) relevant for the purposes of the research described in this paper. The motivation behind the choice of the two formalisms is stimulated by the following: complementary properties of the formalisms [9], [10], current research in field of formal methods integration [11], [12] and the availability of the transformation tools [13], [10], [14].

By the complementary properties of the considered formalisms we mainly mean the following. While both the states and the actions of the considered system are precisely described in a Petri net model, process algebraic specification is usually more focused on describing its dynamic behavior, without explicit representation of states. This also has an impact on the analytical techniques available for the particular formalism. On the other hand, the decomposition of Petri net specifications in general is not so natural as in case of process algebraic specifications.

Although the properties of considered transformations have been formally investigated in some of our work (e.g. [10], [12]), we believe it would also be interesting to validate their implementations practically, using several experiments. This would provide as with valuable knowledge whether the transformation tools adhere precisely enough to definitions of transformations, or maybe the transformations themselves would be updated in some respect. So the following research questions are discussed within this work:
RQ1: What new analytical possibilities would become available after the transformation?
RQ2: How well is preserved the system description after performing the transformation in both directions (reverse transformation)?

2 Formal Methods in Use

Petri net is a powerful mathematical and graphical modeling tool applicable to systems of various and diverse focus. Generally, Petri nets are used for description and analysis of concurrent, asynchronous, distributed, non-deterministic or stochastic systems [15]. According to formal definition, Petri net is a bipartite directed graph populated by three types of objects. These objects are places, transitions, and directed arcs. Directed arcs connect places to transitions or transitions to places. The ability to study the dynamic behavior of a Petri net modeled system in terms of its states and state changes is connected to occurrence of tokens in places. Each place may potentially hold either none or a positive number of tokens [16]. Petri net N can be mathematically defined as arranged 5-

tuple $N = (P, T, pre, post, M_0)$ [17], [18]. In this paper, we consider the concept of ordinary Petri nets, which means that the weight of every arc in Petri net is 1. The execution of Petri net is represented by the flow of tokens in given net and flow of tokens is driven by the *enabling rule* and *transition rule* [16].

Petri nets as a mathematical tool can be characterized by several properties. These properties can by categorized into two groups – *behavioral* and *structural* [15]. Behavioral properties depend on the initial marking, while the structural ones are derived from the topological structure of Petri nets and are independent of some concrete initial marking. Behavioral properties include reachability, boundedness, liveness, reversibility and home state, coverability, persistence, synchronic distance, and fairness [15]. On the other hand, basic structural properties include structural liveness, structural boundedness, conservativeness, repetitiveness, consistency, and S- and T- invariants [17], [15]. In order to gather important insights about modeled system we need to analyze the Petri net model. In general, there are three common approaches to Petri net analysis: 1) the coverability (reachability) tree, 2) the matrix-equation approach, and 3) reduction or decomposition techniques. To help us analyze Petri net models there are several software tools available such as TINA [19], TAPAAL [20], PIPE [21], WoPeD [22] and many others.

Table 1
Axiom system of process algebra ACP

$x + y = y + x$	$(x + y) \between z = x \between z + y \between z$
$(x + y) + z = x + (y + z)$	$ax \mid b = (a \mid b)x$
$x + x = x$	$a \mid bx = (a \mid b)x$
$(x + y)z = xz + yz$	$ax \mid by = (a \mid b)(x \parallel y)$
$(xy)z = x(yz)$	$(x + y) \mid z = x \mid z + y \mid z$
$x + \delta = x$	$x \mid (y + z) = x \lfloor y + x \mid z$
$\delta \cdot x = \delta$	$\partial_H(a) = a \text{ if } a \notin H$
$x \parallel y = x \between y + y \between x + x \lfloor y$	$\partial_H(a) = \delta \text{ if } a \in H$
$a \between x = ax$	$\partial_H(x + y) = \partial_H(x) + \partial_H(y)$
$ax \between y = a(x \parallel y)$	$\partial_H(xy) = \partial_H(x) \cdot \partial_H(y)$

Process algebra is a mathematical framework in which the behavior of the system is expressed in the form of algebraic concepts. The process refers to system behavior. It can be perceived as a summary of discrete actions that the system can perform, the order in which they can occur, but may take into account also other aspects of implementation, such as the timing or likelihood. However, we always describe only certain aspects of behavior. This means that it is an abstraction of the real behavior of the system. Algebra refers to the choice of algebraic (axiomatic) approach in describing behavior [23].

Within this paper we focus on one of the most popular approaches – ACP (Algebra of Communicating Processes) [24] [25]. Process algebra ACP represents algebraic framework for the study of concurrent communicating processes which emphasizes the algebraic aspect. In terms of syntax, process algebra ACP contains a set of constants *A*, special constant δ (deadlock) and operators + (alternative composition), · (sequential composition), || (parallel composition), $\between$(left merge), | (communication). The axiom system of process algebra ACP can be found in Table 1.

When dealing with more complex algebraic specifications, usually the best way to analyze them is by using some of available tools. There are many tools available for analysis of process algebra specifications, but each supporting usually only one (or few) specific formalism (for example PSF Toolkit/ACP [26], mCRL2 Toolset/ACP [27], FDR/CSP [28], PEPA Workbench/PEPA (Performance Evaluation Process Algebra) [29]).

3 Transformation as a Tool for Integration of Formal Methods

Applying different formal methods and different verification techniques can be helpful when using formal methods within the design and analysis of real-life sized systems. The reasons may include that a particular formal technique is the most appropriate for individual components of designed system, the designer wishes to investigate different system properties, or just to manage the complexity of the system [30]. There are several approaches to the integration of different formal methods [17]. The approach employed in this work is based on transformation as tool for integration of formal methods, namely Petri nets and process algebra ACP. In order to support the effectiveness of formal methods integration, we need to choose formalisms whose properties are complementary in some respects. In [10] we mentioned several aspects in which Petri nets and process algebra ACP can be considered mutually complementary.

To perform transformations on selected formal specifications we use several of our software tools. In the process of transformation of algebraic ACP specification to Petri net one PATool and ACP2Petri are used. The main feature of the PATool [14] used here is its ability to convert formats used by external tools. The tool thus represents an interface to multiple process algebra notations. At the moment, it provides standard text editor and format conversions, supporting formats CSP, ACP textual, APC textual and PAML. In context of this research, we use the PATool for conversion of ACP textual format (ACP TXT) to XML-based PAML format [31].

The ACP2Petri tool performs transformation of source algebraic ACP specification in PAML format into the resulting Petri net represented in PNML format [31]. The tool was implemented using the Java programming platform and it has the intuitive graphical user interface. It allows stepping through the process of transformation by individual elementary actions performed during the transformation process and provides also the functionality for exporting the Petri net in PNML and PNG formats. The tool also allows modifying the resulting Petri net layout. Principles, limitations, and further information on this tool can be found in [31] and [13]. The theoretical principles of construction of a Petri net are based on the composition of elementary nets, which represent individual atomic actions of algebraic ACP specification. The composition of elementary nets is performed on the basis of net operations, which correspond to the individual operators of process algebra ACP. A detailed description of Petri net composition rules corresponding to a given ACP term can be found in [32].

For transformation of a Petri net to corresponding algebraic ACP specification we use Petri2ACP tool. The tool is based on research described in [32] and represents a command-line software which transform initial Petri net in PNML format into the resulting algebraic ACP specification represented in PSF format [26]. The theoretical principles behind the transformation implemented within the Petri2ACP tool can be found in [10] and [32].

We also developed another transformation (Petri2APC [33]) which translates a Petri net into an original process algebra APC (Algebra of Process Components) specification, but it is not used in this practical examination. Compared to Petri2ACP transformation it often provides simpler resulting algebraic specifications since in APC there are special constructs available for modeling the synchronization of processes. Currently, there is no tool support for analysis of APC specifications, which limits its practical applications.

3.1 Transformation of Algebraic ACP Specification to Petri Net

Transformations from algebraic ACP specification to Petri net model are evaluated on two simple specifications (Experiment 1 and Experiment 2). Each of these two experiments is described using algebraic process specification (text form of algebraic ACP specification is used within the paper), resulting Petri net model exported using ACP2Petri and results of analysis of corresponding Petri net. For purposes of analysis of resulting Petri nets, several analytical tools can be used, e.g. TAPAAL, TINA, PIPE or WoPeD.

3.1.1 Experiment 1 (MathOP)

In this experiment we start with an algebraic ACP specification of a simple system [34] that performs the calculation of the expression $a^2 + b^2$, which consists of two partial, parallel, and independent computations: $a^2 = a * a$ and $b^2 = b * b$. Subsequently, the addition operation can be performed only when the values of the given partial calculations are available. The algebraic representation of the system follows:

```
A = mula
B = mulb
C = (A || B).add
```

Using the conversion function of the PATool, we can convert the text representation of specification into PAML format, which is then used as an input for ACP2Petri tool (Figure 1) to transform algebraic ACP specification to corresponding Petri net (Figure 2) in PNML format.

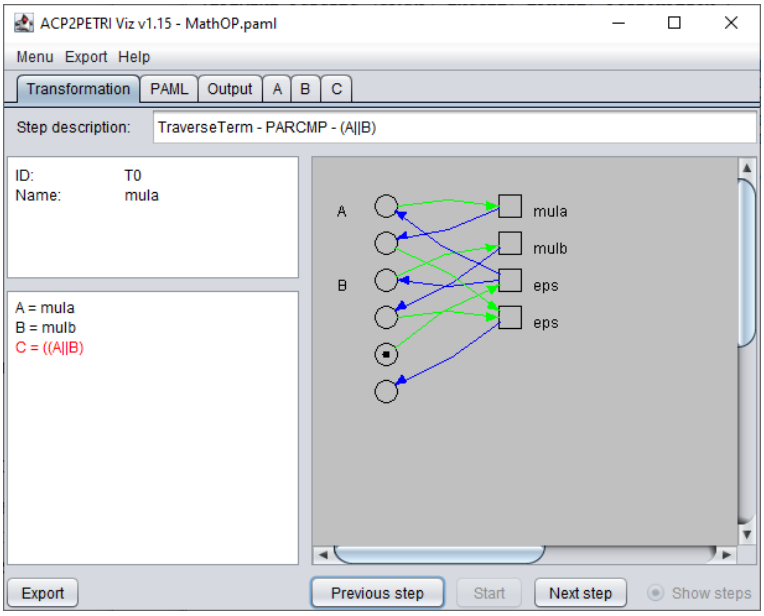


Figure 1
Process of transformation using the ACP2Petri tool

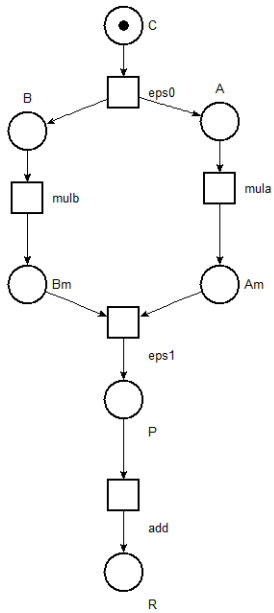


Figure 2
Petri net model of expression calculation corresponding to given algebraic ACP specification

Following the process of transformation, we performed the analysis of Petri net model shown in Figure 2, with the results:

- the net contains 12 nodes (7 places, 5 transitions) and has initial marking M_0(1 0 0 0 0 0 0)
- the net is bounded, safe, contains a deadlock,
- the net has 7 different reachable states,
- the net is not covered by positive T- invariants and is covered by 2 positive S- invariants.

In the case of process algebra ACP we can distinguish between successful and unsuccessful termination (deadlock) [24]. There is no such possibility to distinguish the type of termination in Petri nets used in this research, so the deadlock in the case of resulting Petri net simply means the termination of calculation here.

3.1.2 Experiment 2 (DoubleBuffering)

The next experiment [34] illustrates a general principle of operation of two buffers (primary and secondary) in a process called double buffering, which is used to effectively render graphic information on an output device. While the content of primary buffer is drawn on displaying device, a new graphic information is written

into secondary buffer by graphic hardware. When both operations are finished, the roles are switched. The primary buffer becomes secondary, the secondary one becomes primary and the process repeats. The algebraic specification is given below.

```
gamma (b1ready,b2ready) = ready
encset[H] (b1ready,b2ready)
B1 = draw.b1ready.B1
B2 = read.b2ready.B2
S = encaps[H] (B1||B2)
```

Within the above specification, the first line expresses communication between two actions (`b1ready`, `b2ready`). The second line defines encapsulation set. The next two lines express definitions of processes B1 and B2 and the last line shows the composition of overall process. Again, using conversion capabilities of PATool, we can convert the text representation into PAML format, which represents an input format of the ACP2Petri tool. Next, using the ACP2Petri, we can transform the PAML file to Petri net model (Figure 3).

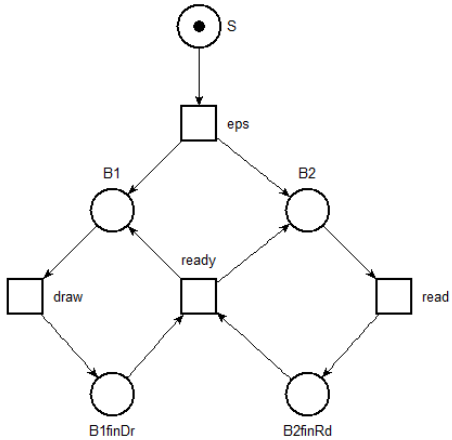


Figure 3

Petri net model of double buffering system corresponding to algebraic ACP specification

After the transformation, we are able to perform analysis of resulting Petri net by means of above-mentioned analytical tools. Results of analysis of double buffering system represented by Petri net model in Figure 3 can be summarized as follows:

- the net contains 9 nodes (5 places, 4 transitions) and has initial marking M_0(1 0 0 0 0),
- the net is bounded, safe and does not contain a deadlock,
- the net has 5 different reachable states,

- the net is not covered by positive T- invariants and is covered by 2 positive S- invariants.

More details on analysis of resulting Petri nets can be found in [34].

3.2 Transformation of Petri Net to Algebraic ACP Specification

Since Petri nets are commonly used to model finite-state machines, dataflow computations [35], communication protocols, multiprocessor systems, etc. [15] we chose to transform a Petri net model of dataflow (Figure 4) and a simple communication protocol (Figure 6) to algebraic ACP specification. The Petri2ACP tool can be used here that performs transformation of Petri net model (PNML format) to algebraic ACP specification (PSF format).

Within the Petri2ACP tool a PNML Framework [36] is used for parsing input PNML files. Although there are several tools supporting PNML interchange format, their PNML variants can differ slightly. We found that while TINA PNML files can be processed by Petri2ACP directly quite well (when using basic features only), WoPeD or TAPAAL PNML files can be processed after some small edits.

In the following two experiments the analysis of resulting algebraic specifications (in PSF format) was performed using the PSF Toolkit. Analytical possibilities of the toolkit are slightly limited, but we believe, they are sufficient to be used here.

3.2.1 Experiment 3 (Dataflow)

The model of dataflow computation system [37] can be expressed by the following pseudocode:

```
BEGIN
  REAL X,Y,R,S;
  X = A + B;
  Y = A - B;
  R = X * Y;
  S = X / Y;
END
```

Next, the dataflow computation expressed by the above pseudocode is represented by the following Petri net model (Figure 4).

In the Petri net (Figure 4) markings in places indicate data availability. Transitions `add`, `sub`, `mul` and `div` represent various operations performed over the actual data values. Operations of addition/subtraction, and multiplication/division respectively, are independent of each other. Depending on data availability in

variables `A` and `B` operations of addition and subtraction can be performed at the same time. The same applies to operations of multiplication and division, if data are available in `X` and `Y`.

Algebraic ACP specification in PSF format can be found in Appendix A at the end of this paper. The analysis of this algebraic specification will focus on the sequence of steps and available executable atomic events and processes at some point of execution. In Figure 5 we can see a simulation of process `Sys`, representing overall composition of processes in algebraic ACP specification. Simulation was performed using the *sim* function of the PSF Toolkit.

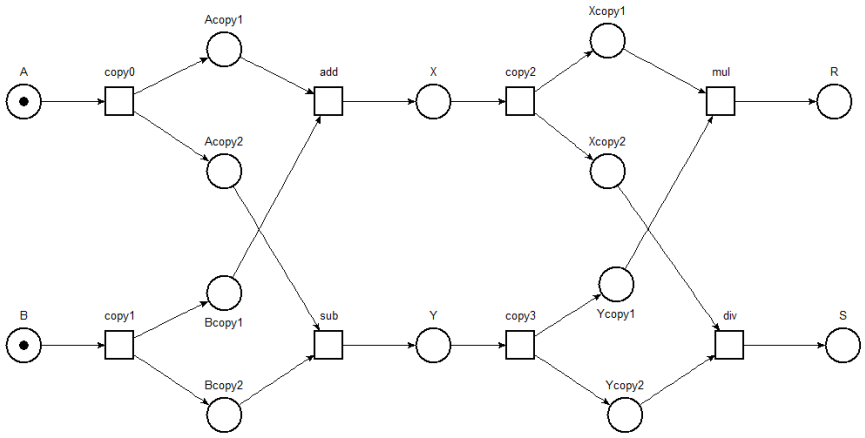


Figure 4
Petri net model of dataflow computation

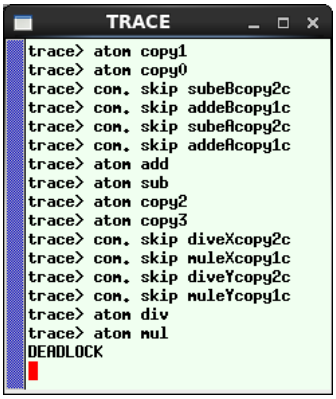


Figure 5
Simulation of algebraic ACP specification of dataflow computation system

Within the process of transformation, some of the names were slightly modified, but we believe that the simulation of the system behavior (Figure 5) is still clear. The inability to perform further actions (after completing the computation) by the Petri net model of simple dataflow computation means that the Petri net contains a deadlock. This state can be reached for example by executing the following sequence of actions: `copy0`, `copy1`, `add`, `copy2`, `sub`, `copy3`, `div`, `mul`. The behavior is preserved also by the corresponding ACP specification.

3.2.2 Experiment 4 (CommProtocol)

Using Petri nets we can also clearly specify the main characteristics of the communication protocols [38]. The properties like liveness and safeness of a Petri net are usually used as criteria for evaluation of communication protocols. The Petri net shown in Figure 6 is a simplified model of a communication between two processes [15].

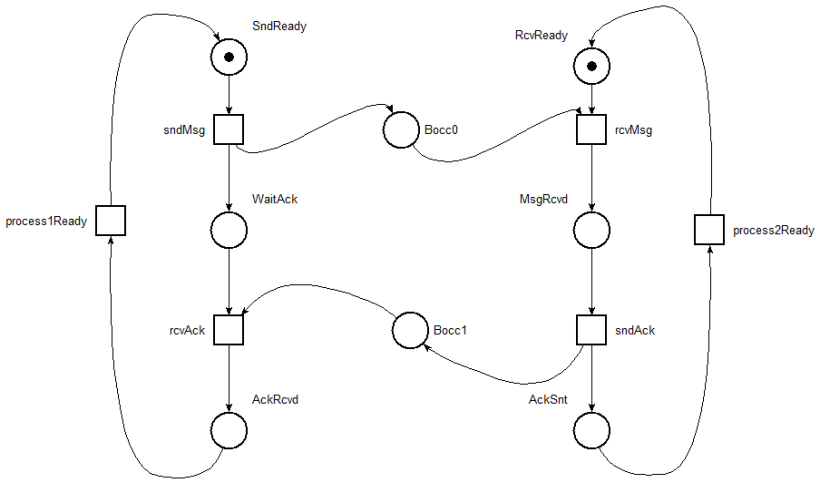


Figure 6
Petri net model of simple communication protocol [15]

Short description of Petri net model from the Figure 6 can be found in Table 2.

The resulting algebraic specification in PSF format of a simplified communication protocol, corresponding to the Petri net in Figure 6, can be found in Appendix B at the end of the paper. The results of simulation of process `Sys` using the simulation functionality of PSF Toolkit, which represent the overall composition of processes in algebraic ACP specification can be seen in Figure 7.

Table 2
Description of communication protocol Petri net model

	Object name	Object type	Description
Sender (Process 1)	SndReady	Place	Information ready to send
	sndMsg	Transition	Sending information
	WaitAck	Place	Waiting for acknowledgement
	rcvAck	Transition	Receiving acknowledgement
	AckRcvd	Place	Acknowledgement received
	process1Ready	Transition	Get ready to send information
medium	Bocc0	Place	Buffer occupied
	Bocc1	Place	Buffer occupied
Receiver (Process 2)	RcvReady	Place	Receiver ready to receive information
	rcvMsg	Transition	Receiving information
	MsgRcvd	Place	Information received
	sndAck	Transition	Sending acknowledgement
	AckSnt	Place	Acknowledgement sent
	process2Ready	Transition	Get ready to receive information

The process does not contain a deadlock therefore it leads to its re-execution, after the second execution of the `sndMsg` action. Also, in this case we can see some composite names as they were constructed in the process of transformation.

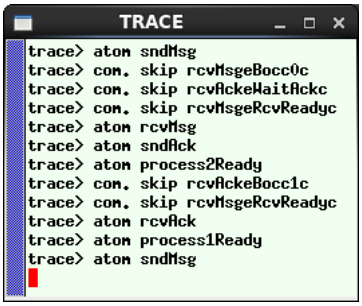


Figure 7
Simulation of algebraic ACP specification for communication protocol model

3.3 Reverse Transformations

The motivation behind reverse transformations is the practical validation of ACP2Petri and Petri2ACP tools used for transformations of Petri net models and algebraic ACP specifications. The goal is to verify the behavior and properties of the system by comparing the initial specification and the specification produced after the reverse transformation. Within this comparison we are interested in

comparing the possible behavior (e.g. sequences of performed actions) of the system, rather than solely syntactical similarity of specifications.

In case of reverse transformation in direction algebraic ACP specification – Petri net – algebraic ACP specification (APA), we are dealing with different formats of algebraic ACP specifications. We start with textual ACP format (and converting it into PAML format using PATool), going through the intermediate PNML specification after ACP2Petri transformation, and finally receive the algebraic specification in PSF format after Petri2ACP transformation.

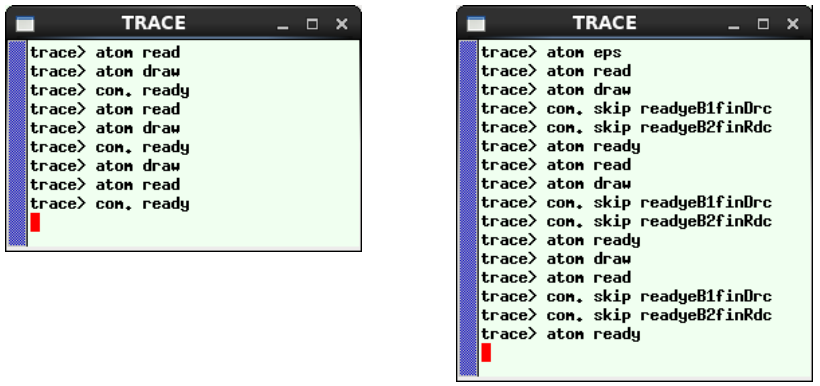
We performed reverse transformations of this type using specifications (Experiment 1 and Experiment 2) from the section 3.1 of the paper. Resulting algebraic specifications in PSF format can be found as Appendix C and Appendix D, respectively. PSF specifications seem to be more complicated, compared to corresponding textual ACP specifications. This may have several reasons, first of them being a fact that PSF files have a dedicated structure with sections like `atoms`, `processes`, `sets of atoms`, etc. The second reason is more profound, as it is connected with transformations used. For example, synchronization of processes in Petri2ACP transformation is provided by defining new auxiliary synchronization process (e.g. `wseps1` in case of specification in Appendix C), corresponding communication functions, set of actions (I) to hide (rename to internal action), and set of actions (H) to encapsulate (rename to δ). More details on principles and properties of Petri2ACP transformation can be found in [10]. So in the end, more atomic actions and more processes can be present in PSF specification as in corresponding textual ACP specification. But since we are interested in behavior described by corresponding algebraic specifications, we used the PSF Toolkit again. In Figure 8 are depicted simulations of specification manually rewritten to PSF format from Experiment 1 (MathOP) (case a)) and specification resulting from reverse transformation (case b)). Actions `eps0` and `eps1` are special actions representing empty process (ε) [39].

a) b)

Figure 8

Simulation of algebraic ACP specifications based on Experiment 1 (MathOP)

Similarly, the results of simulations for the system specified in Experiment 2 (DoubleBuffering) are available in Figure 9. The simulation of specification manually rewritten to PSF format is on the left side (case a)) and the simulation of specification after reverse transformation on the right side of the figure (case b)).



a) b)

Figure 9

Simulation of algebraic ACP specifications based on Experiment 2 (DoubleBuffering)

Now, we would like to discuss the reverse transformation in opposite direction Petri net – algebraic ACP specification – Petri net (PAP). The transformation in this direction is more demanding and restrictive due to difficulty of (currently) manual conversion between PSF format and textual ACP format. Hence, the reverse transformation was performed on a simple concurrent composition of processes, for which the Petri net model is available in Figure 10.

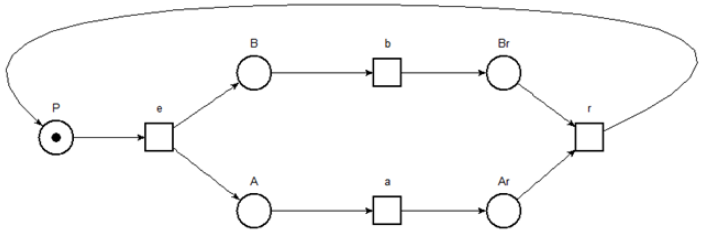


Figure 10

Petri net model of concurrent processes [34]

After performing the first part of this reverse transformation (which is the Petri2ACP transformation) we have gained the algebraic specification in PSF format (Appendix E). The simulation in PSF Toolkit environment confirmed the expected orderings of executed actions (Figure 11).

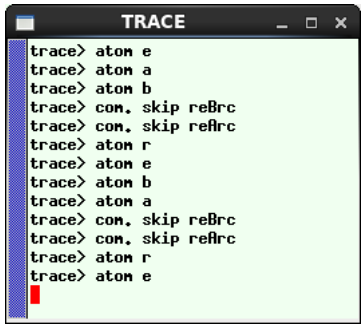


Figure 11
The simulation of simple concurrent system in PSF Toolkit environment

Within the next step, the manual rewriting the PSF specification into the textual ACP format, we initially intended to preserve the original meaning of the PSF specification, where the synchronization of processes is expressed using an auxiliary process (`Wsr`). The ACP specification is available in Table 3, case a).

However, after converting this specification using PATool (into PAML format) and transforming to Petri net using the ACP2Petri tool, the resulting Petri net was too complicated and except the execution of expected actions, using the simulation we were able to disclose also the possibility of calculation termination (deadlock), while the original Petri net and the corresponding PSF specification did not exhibit such possibility.

Table 3
ACP specifications of simple concurrent system

a)	b)

`gamma(ars,arp) = arc`
`gamma(brs,brp) = brc`
`encset[H] (brs,brp,arp,ars)`
`tauset[I] (brc,arc)`
`P = e . (A||B)`
`A = a . Ar`
`B = b . Br`
`Ar = arp`
`Br = brp`
`Wsr = (ars||brs) . r .`
`(P||Wsr)`
`Sys =`
`tau[I](encaps[H](Wsr||P))`
	`P = e . (A||B) . r . P`
	`A = a`
	`B = b`

After this experience we prepared also more simplified version of the specification in textual ACP format (Table 3, case b)) and transformed it into Petri net (Figure 12). By simulation of this Petri net using TINA environment we have confirmed

the expected orderings of actions (`e`, `a`, `b`, and `r`) without the possibility of deadlock.

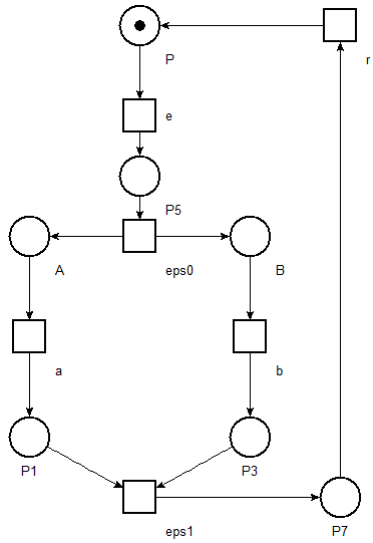


Figure 12
Petri net model of concurrent system after reverse transformation

4 Results and Discussion

Transformations of algebraic ACP specifications to Petri Nets, realized using the PATool and ACP2Petri tools, can be considered convenient and efficient. The advantages in our opinion include the possibility of preparing an algebraic specification in compact ACP textual format and converting it into XML-based PAML format using the PATool, intuitive graphical user interface and overall functionality of the tools. It is necessary to highlight the great contribution that the transformation provides in the field of system analysis. The resulting PNML format is generally supported by many Petri net tools. Therefore, it is possible to perform an extensive and detailed analysis of the considered system and its behavior.

In case of transformations of Petri nets into algebraic ACP specifications, the Petri2ACP tool was used. One of the biggest advantages of algebraic specifications is the support for de/composition, so we are able to explore individual processes, which can be especially useful in case of more complicated systems. However, there have been also several limitations we would take into account. PNML files produced by some of Petri net tools cannot be processed by

the Petri2ACP tool directly. We found that PNML files produced for instance by TINA [19] were generally accepted well by the tool. For some users, the absence of a graphical user interface can be considered a disadvantage, too. And currently, only one output format (PSF) is supported by the tool. Also, there have been several complications while installing the PSF Toolkit. Thanks to the willing support of PSF Toolkit administrator and his feedback, we were able to resolve some of the issues and use the toolkit for the purposes of this research. We appreciated the simulation capabilities of the toolkit. So the transformation of Petri nets to algebraic specifications using the Petri2ACP tool can be considered less comfortable in some respects.

The answer to the RQ1 thus depends on the transformation performed. Using the ACP2Petri transformation we can gain the access to many Petri net tools and their strong analytical capabilities, based on structural analysis, state space analysis, model checking, or reachability graphs. Some of those capabilities are quite unique, like computation of S- and T- invariants. In the case of the Petri2ACP transformation we mentioned the support for de/composition of specification to individual processes, which, in our opinion, can be done in more natural way with process algebra. Also, a simple possibility of renaming actions to internal ones can be useful when investigating behavior of systems at different levels of abstraction [12].

We also performed a practical validation of both transformation tools (ACP2Petri and Petri2ACP) by the means of reverse transformations. In the case of reverse APA transformation we were quite satisfied with the results we achieved. There is a small complication, however. Since there is currently no conversion option to PSF format in PATool, we were forced to rewrite the source ACP specifications into PSF format to be able to compare them with the resulting algebraic specifications using the simulations in the PSF Toolkit environment.

In the case of the reverse PAP transformation we needed a manual conversion from the PSF format (resulting from the first step of PAP transformation) into textual ACP format. Then, after automated conversion into PAML format we were able to perform the second part of the reverse PAP transformation using the ACP2Petri tool. Using the simulations we found a possibility of deadlock in resulting Petri net, while the original Petri net and the corresponding PSF specification did not exhibit such possibility. This finding is worthy of further investigation.

So the definitive answer to the RQ2 would require additional research. On the one hand the results of the APA reverse transformation are quite promising. On the other hand, based on the results of the PAP reverse transformation, there is a possibility of unwanted behavior in the ACP2Petri transformation. We would like to investigate this issue further, as we would like to keep the original behavior, including the possibility of deadlock also after the transformation. Also, there are

some limitations given by the requirement of manual conversions between formats of algebraic specifications.

Conclusions

Based on results of this work, transformations of formalisms with complementary properties like Petri nets and process algebra ACP represent powerful and effective tool in field of integration of formal methods. Combination of formal methods allows us to explore the different aspects of system behavior. Since we only used specifications of relatively modest size in our experiments, there is still a possibility that some benefits or imperfections of considered transformations and associated tools were not fully revealed. Even if the results may not be exactly as we would like to see at the beginning in every aspect, they are very valuable in order to stimulate our further research in this field.

In the future we would like to investigate in deeper detail the possible issue in the ACP2Petri transformation implementation. As it was mentioned above, we would like to preserve the possibility of deadlock behavior before and after the transformation. When comparing our integration method to other approaches combining Petri nets and process algebra we consider the availability of corresponding software tools to be a real benefit supporting its practical application. So it is also our priority to keep the transformation tools in good condition. Another possibility is also an extension of the PATool's conversion capabilities. The most obvious would be the support for PSF format, but we may also consider other formats, like mCRL2 [27] in order to extend the available analytical options for algebraic specifications.

It would be also interesting to explore the possibilities of involving in transformations higher-level Petri nets [19] [20] and corresponding process algebras [40]. Incorporating also the concept of time into the transformations would provide the support for studying time-critical systems [41].

References

[1] J. Woodcock, P. Gorm Larsen, J. Bicarregui, J. Fitzgerald: Formal methods: Practice and experience. ACM Computing Surveys, Vol. 41, No. 4, October 2009

[2] A. Funes, A. Dasso: Formal methods overview. Encyclopedia of Information Science and Technology, 3rd Ed., IGI Global, 2014, pp. 7152-7161

[3] W. Steingartner: On some innovations in teaching the formal semantics using software tools. Open Computer Science, Vol. 11, Issue 1, 2020, pp. 2-11, https://doi.org/10.1515/comp-2020-0130

[4] C. Rouff, M. Hinchey, J. Rash, W. Truszkowski, D. F. Gordon-Spears, (Eds.): Agent Technology from a Formal Perspective. London: Springer-Verlag, 2006

[5] E. M. Clarke, J. M. Wing: Formal Methods: State of the Art and Future Directions. ACM Computing Surveys, Vol. 28, Issue 4, Dec. 1996, pp. 626-643, https://doi.org/10.1145/242223.242257

[6] R. Boute: Integrating Formal Methods by Unifying Abstractions. Integrated Formal Methods, 4th International Conference, IFM 2004, LNCS 2999, Springer-Verlag, 2004

[7] W. Grieskamp, T. Santen, B. Stoddart, (Eds.): Integrated Formal Methods, Second International Conference, IFM 2000 Dagstuhl Castle, Germany, November 1-3, 2000, Lecture Notes in Computer Science, Vol. 1945, Springer-Verlag, 2000

[8] E. A. Boiten, J. Derrick, G. Smith, (Eds.): Integrated Formal Methods, 4th International Conference, IFM 2004, Lecture Notes in Computer Science 2999, Springer-Verlag, 2004

[9] T. Basten: In Terms of Nets: System Design With Petri Nets and Process Algebra. Eindhoven University of Technology, 1998

[10] S. Šimoňák, M. Tomášek: ACP Semantics for Petri Nets. Computing and Informatics, 2018, 37(6), pp. 1464-1484

[11] R. Gorrieri: Language Representability of Finite P/T Nets. In: Programming Languages with Applications to Biology and Security, Volume 9465 of the series Lecture Notes in Computer Science, pp. 262-282, Springer International Publishing, 2015

[12] S. Šimoňák, et al.: Abstraction-enriched Formal Methods Integration. Acta Polytechnica Hungarica, Budapest, Óbuda University, Vol. 15, No. 7, 2018, pp. 179-200

[13] S. Šimoňák, M. Šolc: Enhancing Formal Methods Integration with ACP2Petri. Journal of Informational and Organizational Sciences, Vol. 40, No. 2, 2016, pp. 221-235

[14] S. Šimoňák, I. Peťko: PATool – A tool for design and analysis of discrete systems using process algebras with fdt integration support. Acta Electrotechnica et Informatica, Vol. 10, No. 1, 2010, pp. 59-67

[15] T. Murata: Petri-Nets: Properties, Analysis and Applications. In: Proceedings of the IEEE, Vol. 77, No. 4, 1989

[16] J. Wang: Petri Nets for Dynamic Event-Driven System Modeling. In: Handbook of System Modeling. West Long Branch (New Jersey): Monmouth University, Department of Software Engineering, 2007

[17] S. Šimoňák: Modeling and analysis of systems based on formal methods integration. habilitation thesis, TU FEEI, DCI, Košice, Slovakia, 2018 (in Slovak)

[18] A. Halder: A Study of Petri Nets: Modeling, Analysis and Simulation. Department of Aerospace Engineering, Indian Institute of Technology Kharagpur, August 2006

[19] B. Berthomieu, F. Vernadat: Time Petri Nets Analysis with TINA. tool paper, In: Proceedings of 3rd Int. Conf. on The Quantitative Evaluation of Systems (QEST 2006), IEEE Computer Society, 2006

[20] A. David, L. Jacobsen, M. Jacobsen, K. Y. Jørgensen, M. H. Møller, J. Srba: TAPAAL 2.0: Integrated Development Environment for Timed-Arc Petri Nets. In: Flanagan C., König B. (eds.) Tools and Algorithms for the Construction and Analysis of Systems. TACAS 2012. Lecture Notes in Computer Science, Vol. 7214, Springer, Berlin, Heidelberg

[21] N. J. Dingle, W. J. Knottenbelt, T. Suto: PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets. ACM SIGMETRICS Performance Evaluation Review (Special Issue on Tools for Computer Performance Modelling and Reliability Analysis), Vol. 36(4), March 2009, pp. 34-39

[22] T. Freytag, P. Allgaier, A. Burattin, A. Danek-Bulius: WoPeD – A "Proof-of-Concept" Platform for Experimental BPM Research Projects. In: Proceedings of the BPM Demo Sessions, Barcelona/Spain, September 2017

[23] W. Fokkink: Introduction to Process Algebra. Computer Science – Monograph (English), 2nd edition, Springer-Verlag, 2007

[24] J. C. M. Baeten, W. P. Weijland: Process Algebra. Cambridge University Press, 1990

[25] R. De Nicola: A gentle introduction to Process Algebras. IMT – Institute for Advanced Studies Lucca, Italy, 2013

[26] B. Diertens: Software Engineering with Process Algebra. Ph.D. Thesis, University of Amsterdam, 2009, http://staff.fnwi.uva.nl/b.diertens/phd/

[27] J. F. Groote, J. J. A. Keiren, B. Luttik, E. P. de Vink, T. A. C. Willemse: Modelling and Analysing Software in mCRL2. FACS 2019, LNCS Vol. 12018, pp. 25-48, https://doi.org/10.1007/978-3-030-40914-2_2

[28] T. Gibson-Robinson, P. Armstrong, A. Boulgakov, A. W. Roscoe: FDR3 — A Modern Refinement Checker for CSP. In: E. Ábrahám, K. Havelund (eds.) Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2014, Lecture Notes in Computer Science, Vol. 8413, Springer, Berlin, Heidelberg, https://doi.org/10.1007/978-3-642-54862-8_13

[29] A. M., Abhilash, R. P. Mahapatra: Evaluation of Parallel System using Process Algebra. International Journal of Innovative Technology and Exploring Engineering, 2019, Vol. 8, Issue 9S2, pp. 177-182

[30] E. Ábrahám, M. Huisman (Eds.): Integrated Formal Methods, 12th International Conference, IFM 2016, Reykjavik, Iceland, June 1-5, 2016, Proceedings

[31] S. Šimoňák, Š. Hudák, Š. Korečko: ACP2PETRI: a tool for FDT integration support. In: Proceedings of 8th International Conference EMES´05, 2005, pp. 122-127

[32] S. Šimoňák: Formal Methods Integration Using Transformations of Petri Nets and Process Algebra. Ph.D. dissertation, TU FEEI, DCI, Košice, Slovakia, 2003 (in Slovak)

[33] S. Šimoňák, Š. Hudák, Š. Korečko: APC semantics for Petri nets. in: Informatica, Vol. 32, No. 3, 2008, pp. 253-260

[34] D. Harvilík: Examination of Formal Methods Transformations Properties. master's thesis, TU FEEI, DCI, Košice, Slovakia, 2021 (in Slovak)

[35] B. Madoš, N. Ádám, A. Baláž, K. Šinaľová: The CASE tool for programming of the multi-core System-on-a-Chip with the data flow computation control. In: SAMI 2017. Danvers, IEEE, 2017, pp. 165-168

[36] L. Hillah, F. Kordon, L. Petrucci, and N. Trèves: PNML Framework: an extendable reference implementation of the Petri Net Markup Language. Petri Nets 2010, LNCS 6128, pp. 318-327

[37] S. F. Jennings: Petri net models of program execution in data flow environments. Retrospective Theses and Dissertations, 1981

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

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

[40] O. Bunte, J. F. Groote, J. J. A. Keiren, M. Laveaux, T. Neele, E. P. de Vink, J. W. Wesselink, A. J. Wijs, T. A. C. Willemse: The mCRL2 Toolset for Analysing Concurrent Systems: Improvements in Expressivity and Usability. TACAS 2019, LNCS Vol. 11428, pp. 21-39

[41] Š. Hudák, Š. Korečko, S. Šimoňák: Reachability analysis of time-critical systems. Petri Nets: Applications. Vukovar, In-Tech, 2010, pp. 253-280

Appendix A – algebraic specification (PSF format) of dataflow computation (Experiment 3)

```
process module SYSdataflow
begin
```

```
  atoms
sub,subeBcopy2s,subeBcopy2p,mul,div,subeBcopy2c,muleXcopy1c,muleYcopy
1p,muleYcopy1s,add,muleXcopy1p,addeBcopy1c,subeAcopy2p,muleXcopy1s,di
veXcopy2s,subeAcopy2c,diveYcopy2c,diveXcopy2p,addeAcopy1s,addeAcopy1p
,muleYcopy1c,addeBcopy1s,diveXcopy2c,addeBcopy1p,copy3,subeAcopy2s,di
veYcopy2s,copy2,diveYcopy2p,copy1,addeAcopy1c,copy0
  processes
eBcopy1,eAcopy2,wssub,eAcopy1,wsadd,wsmul,eR,eXcopy2,eS,eXcopy1,eYcop
y2,eYcopy1,eX,eY,wsdiv,eA,eBcopy2,eB,Sys
  sets of atoms
    H =
{muleXcopy1p,subeBcopy2s,subeBcopy2p,subeAcopy2p,muleXcopy1s,diveXcop
y2s,diveXcopy2p,addeAcopy1s,addeAcopy1p,addeBcopy1s,addeBcopy1p,subeA
copy2s,diveYcopy2s,diveYcopy2p,muleYcopy1p,muleYcopy1s}
    I =
{addeBcopy1c,subeBcopy2c,muleXcopy1c,diveXcopy2c,subeAcopy2c,diveYcop
y2c,addeAcopy1c,muleYcopy1c}
  communications
    muleXcopy1s|muleXcopy1p = muleXcopy1c
    diveYcopy2s|diveYcopy2p = diveYcopy2c
    addeAcopy1s|addeAcopy1p = addeAcopy1c
    addeBcopy1s|addeBcopy1p = addeBcopy1c
    muleYcopy1s|muleYcopy1p = muleYcopy1c
    subeBcopy2s|subeBcopy2p = subeBcopy2c
    diveXcopy2s|diveXcopy2p = diveXcopy2c
    subeAcopy2s|subeAcopy2p = subeAcopy2c
  definitions
    eR = delta
    eS = delta
    eAcopy1 = addeAcopy1p
    eYcopy1 = muleYcopy1p
    eAcopy2 = subeAcopy2p
    eXcopy1 = muleXcopy1p
    eXcopy2 = diveXcopy2p
    eB = copy1 . (eBcopy1||eBcopy2)
    eA = copy0 . (eAcopy1||eAcopy2)
    eY = copy3 . (eYcopy2||eYcopy1)
    eX = copy2 . (eXcopy1||eXcopy2)
    eBcopy2 = subeBcopy2p
    eBcopy1 = addeBcopy1p
    eYcopy2 = diveYcopy2p
    wsadd = (addeAcopy1s||addeBcopy1s) . add . (eX||wsadd)
    wsmul = (muleXcopy1s||muleYcopy1s) . mul . (eR||wsmul)
    wssub = (subeAcopy2s||subeBcopy2s) . sub . (eY||wssub)
    wsdiv = (diveYcopy2s||diveXcopy2s) . div . (eS||wsdiv)
    Sys = hide(I,encaps(H,(wsadd||wsdiv||wssub||wsmul||eB||eA)))
end SYSdataflow
```

Appendix B – algebraic specification (PSF format) of simple communication protocol (Experiment 4)

```
process module SYSCommProtocolTINA
begin
  atoms
rcvAckeBocc1s,sndMsg,process1Ready,rcvMsgeRcvReadys,rcvMsgeBocc0c,rcv
Msg,rcvMsgeRcvReadyp,rcvAckeWaitAcks,rcvAckeBocc1c,rcvMsgeBocc0p,rcvA
ckeWaitAckp,rcvAck,process2Ready,sndAck,rcvMsgeRcvReadyc,rcvAckeBocc1
p,rcvMsgeBocc0s,rcvAckeWaitAckc
  processes
eAckSnt,eBocc1,eBocc0,wsrcvMsg,eRcvReady,eAckRcvd,eWaitAck,eSndReady,
wsrcvAck,eMsgRcvd,Sys
  sets of atoms
    H =
{rcvAckeBocc1s,rcvMsgeBocc0p,rcvAckeWaitAckp,rcvMsgeRcvReadys,rcvAcke
Bocc1p,rcvMsgeBocc0s,rcvMsgeRcvReadyp,rcvAckeWaitAcks}
    I =
{rcvAckeBocc1c,rcvMsgeRcvReadyc,rcvMsgeBocc0c,rcvAckeWaitAckc}
  communications
    rcvAckeBocc1s|rcvAckeBocc1p = rcvAckeBocc1c
    rcvMsgeBocc0s|rcvMsgeBocc0p = rcvMsgeBocc0c
    rcvMsgeRcvReadys|rcvMsgeRcvReadyp = rcvMsgeRcvReadyc
    rcvAckeWaitAcks|rcvAckeWaitAckp = rcvAckeWaitAckc
  definitions
    eMsgRcvd = sndAck . (eBocc1||eAckSnt)
    eSndReady = sndMsg . (eWaitAck||eBocc0)
    eRcvReady = rcvMsgeRcvReadyp
    eAckSnt = process2Ready . eRcvReady
    eBocc0 = rcvMsgeBocc0p
    eAckRcvd = process1Ready . eSndReady
    eWaitAck = rcvAckeWaitAckp
    eBocc1 = rcvAckeBocc1p
    wsrcvMsg = (rcvMsgeRcvReadys||rcvMsgeBocc0s) . rcvMsg .
(eMsgRcvd||wsrcvMsg)
    wsrcvAck = (rcvAckeBocc1s||rcvAckeWaitAcks) . rcvAck .
(eAckRcvd||wsrcvAck)
    Sys =
hide(I,encaps(H,(wsrcvAck||wsrcvMsg||eSndReady||eRcvReady)))
end SYSCommProtocolTINA
```

Appendix C – algebraic specification (PSF format) of simple calculation (Experiment 1)

```
process module SYSMathOPrev
begin
```

```
  atoms
add,eps1eBmc,eps1eBms,eps1eAmc,eps1eAms,eps1,eps1eBmp,eps0,eps1eAmp,m
ulb,mula
  processes
    eAm,eBm,wseps1,eP,eA,eR,eB,eC,Sys
  sets of atoms
    H = {eps1eBms,eps1eAms,eps1eBmp,eps1eAmp}
    I = {eps1eBmc,eps1eAmc}
  communications
    eps1eBms|eps1eBmp = eps1eBmc
    eps1eAms|eps1eAmp = eps1eAmc
  definitions
    eR = delta
    eB = mulb . eBm
    eA = mula . eAm
    eBm = eps1eBmp
    eP = add . eR
    eAm = eps1eAmp
    eC = eps0 . (eB||eA)
    wseps1 = (eps1eAms||eps1eBms) . eps1 . (eP||wseps1)
    Sys = hide(I,encaps(H,(wseps1||eC)))
end SYSMathOPrev
```

Appendix D – algebraic specification (PSF format) of double buffering system (Experiment 2)

```
process module SYSDoubleBufferingrev
begin
  atoms
readyeB1finDrp,readyeB1finDrc,readyeB1finDrs,read,ready,readyeB2finRd
c,readyeB2finRds,eps,readyeB2finRdp,draw
  processes
    eB2,eB1,eB2finRd,eB1finDr,eS,wsready,Sys
  sets of atoms
    H = {readyeB1finDrp,readyeB1finDrs,readyeB2finRds,readyeB2finRdp}
    I = {readyeB1finDrc,readyeB2finRdc}
  communications
    readyeB2finRds|readyeB2finRdp = readyeB2finRdc
    readyeB1finDrs|readyeB1finDrp = readyeB1finDrc
  definitions
    eB2finRd = readyeB2finRdp
    eB1 = draw . eB1finDr
    eB1finDr = readyeB1finDrp
    eS = eps . (eB2||eB1)
    eB2 = read . eB2finRd
```

```
    wsready = (readyeB2finRds||readyeB1finDrs) . ready .
(eB1||eB2||wsready)
    Sys = hide(I,encaps(H,(wsready||eS)))
end SYSDoubleBufferingrev
```

Appendix E – algebraic specification (PSF format) of simple concurrent system

```
process module SYSInverznyPrevod-BackTransfTINA
begin
  atoms
    a,r,b,e,reBrc,reBrs,reBrp,reArp,reArc,reArs
  processes
    eBr,eAr,wsr,eP,eA,eB,Sys
  sets of atoms
    H = {reBrs,reBrp,reArp,reArs}
    I = {reBrc,reArc}
  communications
    reArs|reArp = reArc
    reBrs|reBrp = reBrc
  definitions
    eAr = reArp
    eP = e . (eA||eB)
    eA = a . eAr
    eBr = reBrp
    eB = b . eBr
    wsr = (reArs||reBrs) . r . (eP||wsr)
    Sys = hide(I,encaps(H,(wsr||eP)))
end SYSInverznyPrevod-BackTransfTINA
```