Моделирование и анализ информационных систем. Т. 22, №4 (2015), с. 563–577
Modeling and Analysis of Information Systems. Vol. 22, No 4 (2015), pp. 563–577

DOI: 10.18255/1818-1015-2015-4-563-577

UDC 519.681.2

A Method of Sample Models of Program Construction in Terms of Petri Nets

Kharitonov D.I., Golenkov E.A., Tarasov G.V., Leontyev D.V.

Received September 1, 2015

In the article a method of automated construction of Petri nets simulating the behaviour of imperative programs is considered from the formal point of view. Petri net samples with certain characteristics are necessary in programming new algorithms for program analysis; in particular, they can be used for developing or optimizing algorithms of Petri nets compositions and decompositions, building the reachability tree, checking invariants and so on. The generation process consists of two stages. At the first stage, construction templates for a resulting net and parameters for construction are described. With the help of these parameters it is possible to regulate the final size and the absolute or relative amount of certain structures in the resulting net. At the second stage, iterative process of automated net construction is used for Petri net generation of any size, limited only by an available computer memory. In the first section of the article the minimum necessary definitions are given and a new version of Petri nets composition operation by places is introduced. Commutative and associative properties of introduced binary operation allow to synchronize any number of Petri nets in arbitrary order. Then construction template is defined as a marked Petri net with input and output interfaces and rules for templates composition using this interfaces. A number of construction templates can be united in a collection, for which the evolution rules are defined. The completeness property of a collection guarantees that the collection evolution results in a Petri net that simulates the imperative program behavior. The article provides a version of the construction templates complete collection and an example of Petri net simulating sequential imperative program construction.

The article is published in the author's wording.

Keywords: program model, control flow model, Petri net object

For citation: Kharitonov D.I., Golenkov E.A., Tarasov G.V., Leontyev D.V., "A Method of Sample Models of Program Construction in Terms of Petri Nets", *Modeling and Analysis of Information Systems*, **22**:4 (2015), 563–577.

On the authors:
Kharitonov Dmitry Ivanovich, orcid.org/0000-0003-3359-2383, PhD, senior researcher,
Institution of Russian Academy of Sciences Institute of Automation and Control Processes Far Eastern Branch of the RAS, 5 Radio str., Vladivostok, Russia, 690041, e-mail: demiurg@dvo.ru

Golenkov Evgeny Alexandrovich, orcid.org/0000-0002-8148-3504, PhD, senior researcher,
Institution of Russian Academy of Sciences Institute of Automation and Control Processes Far Eastern Branch of the RAS, 5 Radio str., Vladivostok, Russia, 690041, e-mail: golenkov@dvo.ru

Tarasov Georgiy Vitalievich, orcid.org/0000-0001-8855-7388, research officer,
Institution of Russian Academy of Sciences Institute of Automation and Control Processes Far Eastern Branch of the RAS, 5 Radio str., Vladivostok, Russia, 690041,
Far-Eastern Federal University, 8 Suhanova st., Vladivostok, Russia, 690950, e-mail: george@dvo.ru

Leontiev Denis Valerievich, orcid.org/0000-0002-5116-3008, postgraduate student,
Institution of Russian Academy of Sciences Institute of Automation and Control Processes Far Eastern Branch of the RAS, 5 Radio str., Vladivostok, Russia, 690041,
Far-Eastern Federal University, 8 Suhanova st., Vladivostok, Russia, 690950, e-mail: devozh@dvo.ru

Acknowledgments:
This work was supported by the Russian academy of sciences fundamental research program "Fundamental problems of mathematical modelling", project 0262-2014-0157.

Introduction

Automatics and machinery in the modern world more and more relies on software. In many areas of human activity software errors may cost human lives. For example, in 2000 an erroneous calculation of the radiation dose led to several deaths [4]. However, among the variety of existing programs only very few have been formally verified, proving their correctness. This situation caused by the necessity of the human intellect to describe the programs examined in terms suitable to analysis. Petri nets are one of the few formal languages allowing to automate the process of software systems behavior models construction. In some cases, Petri nets are extremely suitable for modeling due to the distributed nature of the systems described, such as the development of multimedia streams scenarios [11]. In other cases, Petri nets analysis tools meet the stated objectives, like in the development of the process managing web services [9]. Software engineering and Petri nets crossed several times in the past resulting in interesting ideas in both areas [6]. Certain steps have been done by the authors of this article towards imperative programs modeling [7,17,18]. Nevertheless, there is a serious concern that the advantages of Petri nets as a formalism for distributed systems description with a clear graphical presentation will be lost, when describing the programs of actual complexity. At first, nets with more than a thousand of elements can not be represented on the screen or on the printed page in a readable form. At second, more significantly, nets analysis algorithms, for example, reachability tree construction algorithm, are to be adapted for Petri nets with a large number of elements. The classic reachability tree construction algorithm [1] for the nets greater than of 10^5 places and transitions requires more than 10Gb of memory that can be considered is a threshold for personal computers. In the international competition “Model Checking Contest @ Petri Net” for the comparison of Petri nets analysis tools a set of predefined models is used, and in 2015 the largest, by the number of places and transitions, model had about 34 thousand elements [20]. This number of elements corresponds to the Petri nets modeling imperative programs of less than 10 thousand lines of code, while the larger software systems can have hundreds of thousands of lines. For the adaptation of the algorithms dealing with Petri nets and their quality investigation there is a demand for the nets with a predefined number of elements and with known properties. The authors concluded that automatic generation of such nets is an important issue.

Material in the article is presented in the next way. The first section provides the minimum of the necessary definitions and a simple Petri nets composition operation by places is introduced. The second section describes the notion of the construction template and defines the rules of Petri nets automatic generation. The third section provides a complete set of construction templates and an example of generation of Petri net simulating behaviour of imperative program. Finally, conclusions on the applicability the method proposed are drawn.

1. Simple Petri net composition by places

Let $A = \{a_1, a_2, ..., a_k\}$ is a set. Multiset on set A is a function $\mu : A \to \{0, 1, 2, \dots\}$, that assigns a non-negative integer to each element of the set A. Multiset is conveniently written as a formal sum $n_1a_1 + n_2a_2 + ... + n_ka_k$ or Σn_ia_i, where $n_i = \mu(a_i)$ is the number of occurrences of the $a_i \in A$ in the multiset. Normally, when recording the sum, its zero elements $n_i = 0$ are omitted. The arithmetical sum and difference of multisets μ_1 and μ_2 are defined, respectively, as

$$(\mu_1 + \mu_2)(a) = \mu_1(a) + \mu_2(a),$$

$$(\mu_1 - \mu_2)(a) = \begin{cases} \mu_1(a) - \mu_2(a), & \text{если } \mu_2(a) \leq \mu_1(a); \\ 0, & \text{otherwise.} \end{cases}$$

Comparing multisets μ_1 and μ_2 it is right to write: $\mu_1 \leq \mu_2$, if $\forall a \in A : \mu_2(a) \leq \mu_1(a)$, and $\mu_1 \geq \mu_2$, if $\forall a \in A : \mu_2(a) \geq \mu_1(a)$. If $n_i = 0$ for all i, then this multiset will be denoted as $\mathbf{0}$. We will also write that $a \in \mu$, if $\exists n > 0 : (a, n) \in \mu$. The set of all finite multisets on the set A will be denoted as $\mathcal{M}(A)$.

Let's define a sequence s on the set A as a function $\mathbb{N}_0 \to A \cup \emptyset$, associating with a positive integer one element of the set A or the empty set element $\emptyset$, if the number is greater than the size of the sequence $|s|$. The sequence is written as $(a_i)_{i=0}^n$ or shorter (a_i). Sequence element a_i is written as the function value of integer argument $s(i)$. The set of all finite sequences in the set A is written as (A). Let's also define a linearly ordered subset B of the set A with a linear order relation $\prec_A$ as $(b_i \mid \forall i < j \leq n \Rightarrow b_i \prec_A b_j)$. Linearly ordered subset is written as $[b_i]_{i=0}^n$ or shortly $[b_i]$. The set of all finite linearly ordered subsets of the set A is written as $[A]$.

Definition 1. *Petri net is a tuple $\Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$, where*

1. *S — a finite set of places;*
2. *T — a finite set of transitions such that $S \cap T = \emptyset$;*
3. *${}^\bullet() : T \to \mathcal{M}(S)$ — input incidence function;*
4. *$()^\bullet : T \to \mathcal{M}(S)$ — output incidence function.*

Multisets ${}^\bullet t$ and $t^\bullet$ are called input and output multisets of transition $t \in T$ accordingly.

Definition 2. *Formal union of Petri nets.*
Let us given two Petri nets $\Sigma_1 = \langle S_1, T_1, {}^\bullet()_1, ()_1^\bullet \rangle$ and $\Sigma_2 = \langle S_2, T_2, {}^\bullet()_2, ()_2^\bullet \rangle$. Formal union of the Petri nets Σ_1 and Σ_2 is the net $\Sigma = \Sigma_1 \oplus \Sigma_2 = \langle S, T, {}^\bullet(), ()^\bullet \rangle$, such that

$$S = S_1 \cup S_2, \qquad T = T_1 \cup T_2.$$

$${}^\bullet(t) = \begin{cases} {}^\bullet(t)_1, & \text{if } t \in T_1; \\ {}^\bullet(t)_2, & \text{if } t \in T_2. \end{cases}$$

$$(t)^\bullet = \begin{cases} (t)_1^\bullet, & \text{if } t \in T_1; \\ (t)_2^\bullet, & \text{if } t \in T_2. \end{cases}$$

Petri nets defined in such a way quite rarely used for modeling real systems, because as the number of places and transitions increases so raises the complexity of model perception as a whole. To simplify modeling of complex systems the compositional approach to build whole model from the simpler models of its subsystems is widely used in practice. The most widely used is a nets composition by transitions [3, 14], but there are variations of nets composition operations by places [10], and also by places and transitions [15]. We introduce the operation of Petri nets composition by places using the scheme proposed in articles [2, 5]. In our case, the goal is to minimize the algorithmic complexity of the operation implementation.

Definition 3. *Simple access point by places to Petri net.*
Let's call the tuple $\iota = \langle id_\iota, \varrho_\iota \rangle$ simple access point by places to Petri net $\Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$, where id_ι — unique identifier of access point, a $\varrho_\iota \in [S]$ — linearly ordered subset of places, used by access point.

The name "simple point of access" is used to distinguish this access point from the ones introduced in the articles [2, 5]. Further in the text instead of the full name "simple access point by places" abbreviation "simple access point" may be used or even just "access point".

Definition 4. *Merge of Petri net simple access points.*
Let us given Petri net $\Sigma_1 = \langle S_1, T_1, {}^\bullet()_1, ()^\bullet_1 \rangle$ and two its simple access points $\iota_1 = \langle id_1, \varrho_1 \rangle$, $\iota_2 = \langle id_2, \varrho_1 \rangle$, such that $|\varrho_1| = |\varrho_2|$ u $\varrho_1 \cap \varrho_2 = \emptyset$. Then merge operation of simple access points ι_1 and ι_2 of Petri net Σ_1 forms new net $\Sigma = \Sigma_1|^{\iota_1}_{\iota_2} = \langle S, T, {}^\bullet(), ()^\bullet \rangle$, so that

1. *$S = S_{const} \cup S_{syn}$, where*
   - $S_{const} = S_1 \setminus (\varrho_1 \cup \varrho_2)$,
   - $S_{syn} = \{\langle \varrho_1(i), \varrho_2(i) \rangle \mid 0 \le i \le |\varrho_1|\}$,

   and there is a mapping surjection between a source and a finite set of places $\Upsilon^{\iota_1}_{\iota_2} : S_1 \to S$, such that
   $$\Upsilon^{\iota_1}_{\iota_2}(s) = \begin{cases} s, & \forall s' \in S_{const} \\ s' = \langle s, \varrho_2(i) \rangle, & \forall s = \varrho_1(i) \in \varrho_1 \\ s' = \langle \varrho_1(j), s \rangle, & \forall s = \varrho_2(j) \in \varrho_2. \end{cases}$$

2. *$T = T_1$,*

3. *$\forall t \in T, s \in S_{const} : {}^\bullet(t)(s) = {}^\bullet(t)_1(s)$ u*
   $\forall t \in T, s = \langle s', s'' \rangle \in S_{syn} : {}^\bullet(t)(s) = {}^\bullet(t)_1(s') + {}^\bullet(t)_1(s'')$,

4. *$\forall t \in T, s \in S_{const} : (t)^\bullet(s) = (t)^\bullet_1(s)$ u*
   $\forall t \in T, s = \langle s', s'' \rangle \in S_{syn} : (t)^\bullet(s) = (t)^\bullet_1(s') + (t)^\bullet_1(s'')$,

5. *$\forall s \in S_{const} : M_0(s) = M_{01}(s)$,*
   $\forall s = \langle s', s'' \rangle \in S_{syn} : M_0(s) = M_{01}(s') + M_{01}(s'')$.

Less formally merge of Petri nets simple access points by places performs "joining" of places, used by the access points, on the principle "one access point place merge another access point place with the same sequence number". Transitions of the original net do not change, and the arcs are restored from the original net, connecting transitions with the mapping of the original incident places. Using a list representation of places, transitions and arcs sets, software implementation of the access points merge operation can be performed along with copying elements from original to destination nets in no more than $O(N)$ CPU operations where N is the number of elements in the original net. The mapping between the source and a finite set of places allows also to convert other simple access points by places not involved in the merge operation.

Definition 5. *Let us given Petri net $\Sigma = \Sigma_1|^{\iota_1}_{\iota_2}$, resulting from the simple access poing merge of the net Σ_1. Then simple access point $\iota = \langle id_\iota, \varrho_\iota\rangle$ by places of net Σ is the convertion of access point $\iota' = \langle id'_\iota, \varrho'_\iota\rangle$ by places of net Σ_1 as the result of merging, if $id'_\iota = id_\iota$ and $\forall i \Rightarrow \varrho_\iota(i) = \Upsilon^{\iota_1}_{\iota_2}(\varrho'_\iota(i))$.*

In practice, two Petri nets merge operation is more frequently used, which is defined as follows.

Definition 6. *Binary Petri nets merge operation by simple access points.
Let us given two Petri nets $\Sigma_1 = \langle S_1, T_1, {}^\bullet()_1, ()^\bullet_1\rangle$, $\Sigma_2 = \langle S_2, T_2, {}^\bullet()_2, ()^\bullet_2\rangle$ and two their simple access points by places $\iota_1 = \langle id_1, \varrho_1\rangle$, $\iota_2 = \langle id_2, \varrho_2\rangle$, such that $|\varrho_1| = |\varrho_2|$. Then Petri nets Σ_1 and Σ_2 merge operation by simple access points ι_1 and ι_2 forms new net $\Sigma = \Sigma_1 \underset{\iota_1\ \iota_2}{\oplus} \Sigma_2 \equiv (\Sigma_1 \oplus \Sigma_2)|^{\iota_1}_{\iota_2}$.*

Software implementation of the binary Petri nets merge operation can be performed, similarly to unary, in no more than $O(N_1 + N_2)$ CPU operations, where N_1 and N_2 are numbers of elements of the original nets. Taking into account the above-described conversion of simple access points by places, let's assume that source net access points are applicable to the net resulting from merging. Then merge operations properties can be written that follow directly from the definitions:

1. Unary operation commutativity

$$\Sigma|^{\iota_1}_{\iota_2} = \Sigma|^{\iota_2}_{\iota_1}$$

   indicates that the result of merging the access point does not depend on the access points order.

2. Binary operation commutativity

$$\Sigma_1 \underset{\iota_1\ \iota_2}{\oplus} \Sigma_2 = \Sigma_2 \underset{\iota_2\ \iota_1}{\oplus} \Sigma_1$$

   allows not to worry about the order of nets in the operation.

3. Unary operation associativity

$$\Sigma|^{\iota_1}_{\iota_2}|^{\iota_3}_{\iota_4} = \Sigma|^{\iota_3}_{\iota_4}|^{\iota_1}_{\iota_2}$$

   allows to perform a number of merge operations over one net in any order.

4. Binary operation associativity

$$\Sigma_1 \underset{\iota_1\ \iota_2}{\oplus} \Sigma_2 \underset{\iota_3\ \iota_4}{\oplus} \Sigma_3 = (\Sigma_1 \underset{\iota_1\ \iota_2}{\oplus} \Sigma_2) \underset{\iota_3\ \iota_4}{\oplus} \Sigma_3 = \Sigma_1 \underset{\iota_1\ \iota_2}{\oplus} (\Sigma_2 \underset{\iota_3\ \iota_4}{\oplus} \Sigma_3)$$

   allows to merge several Petri nets in random order.

2. Construction templates in terms of Petri nets

With the use of simple access points by places to Petri nets and composition operations, introduced in the previous section, we formulate object-oriented approach to the automatic generation of Petri nets. This approach based on the concept of the construction template.

Definition 7. *A template of imperative construction in terms of Petri nets (for short PN-template) is the tuple $X = \langle \Sigma, I, O, M_0 \rangle$, where*

1. *$\Sigma = \langle S, T, {}^\bullet(), ()^\bullet \rangle$ — Petri net, called object structure;*
2. *$I = \{\iota_1, \iota_2, \dots, \iota_n\}$ — the set of simple access points, called input interface;*
3. *$O = \{\phi_1, \phi_2, \dots, \phi_n\}$ — the set of simple access points, called output interface;*
4. *$M_0 \in \mathcal{M}(S)$ — initial marking.*

An imperative construction template is a marked Petri net, having part of the places assigned for merging with “superior” nets as the input interface, and another part of the places - to merge with the “subordinate” nets as the output interface. Let’s formalize construction templates merge operation, “superior” nets are built with the help of.

Definition 8. *Formal union of PN-templates.*
Let us given two imperative construction templates $X_1 = \langle \Sigma_1, I_1, O_1, M_{01} \rangle$ and $X_2 = \langle \Sigma_2, I_2, O_2, M_{02} \rangle$. Formal union of X_1 and X_2 is the template $X = X_1 \oplus X_2 = \langle \Sigma, I, O, M_0 \rangle$, such that

$$\Sigma = \Sigma_1 \oplus \Sigma_2, \qquad I = I_1 \cup I_2, \qquad O = O_1 \cup O_2, \qquad M_0 = M_{01} + M_{02}.$$

Operation of PN-templates formal union makes new template by simple union of the sets and markings of the initial templates. To change the structure of the template the merge operation of simple access points is used.

Definition 9. *Merge of PN-template simple access points.*
Let us given a PN-template $X_1 = \langle \Sigma_1, I_1, O_1, M_{01} \rangle$ and two its simple points $\iota \in I_1, \phi \in O_1$, where $\iota = \langle id_1, \varrho_1 \rangle, \phi = \langle id_2, \varrho_2 \rangle$ and $\Sigma_1 = \langle S_1, T_1, {}^\bullet()_1, ()^\bullet_1 \rangle$. If subsets of places of both simple access points have equal cardinality $|\varrho_1| = |\varrho_2|$, than merge of simple access points operation of imperative construction X_1 by simple access points ι u ϕ forms new template $X = \langle \Sigma, I, O, M_0 \rangle$, where $I = I_1 \setminus \{\iota\}$, $O = O_1 \setminus \{\phi\}$ and $\Sigma = \Sigma_1|^{\iota}_{\phi}$.

Merge of a PN-template simple access points operation (in unary form) is denoted as $X = X_1|^{\iota}_{\phi}$.

More common used, and usefull for us, binary form of templates merge operation is defined by consecutive application of the two above operations.

Definition 10. *PN-templates merge operation by simple access points.*
Let us given two construction templates $X_1 = \langle \Sigma_1, I_1, O_1, M_{01} \rangle$, $X_2 = \langle \Sigma_2, I_2, O_2, M_{02} \rangle$ and two their access points $\iota \in I_1, \phi \in O_2$, where $\iota = \langle id_1, \varrho_1 \rangle, \phi = \langle id_2, \varrho_2 \rangle$. And subsets of places of both simple access points have equal cardinality $|\varrho_1| = |\varrho_2|$. Then templates X_1 and X_2 merge operation by simple access points ι and ϕ forms new template $X = \langle \Sigma, I, O, M_0 \rangle$, so that $X = (X_1 \oplus X_2)|^{\iota}_{\phi}$.

Now it is possible to formulate the necessary requirements to imperative construction templates in order to build program simulations in terms of Petri nets.

Definition 11. *PN-templates collection.*
A set $\Pi = \{X_i = \langle \Sigma_i, I_i, O_i, M_i \rangle\}$ *is called PN-templates collection, if:*

1. *There is the start template* $X_o \in \Pi$, *such that* $M_o > \boldsymbol{0}, |I_o| = 0, |O_o| \geq 0$.
2. *There are building templates:* $|\{X_k \mid X_k \in \Pi, |I_k| = 1, |O_k| > 0\}| \geq 1$.
3. *All simple access point in the input interfaces has a pair in the output interfaces, and vice versa:*
   - $\forall X_i \in \Pi, \phi \in O_i \to \exists X_j \in \Pi, \iota \in I_j : |\phi| = |\iota|$,
   - $\forall X_i \in \Pi, \iota \in I_i \to \exists X_j \in \Pi, \phi \in O_j : |\phi| = |\iota|$.

Practically, a templates collection - is a system in which the result of the start template merging with any of the others gives a new start template.

Definition 12. *PN-templates collection evolution.*
PN-templates collection $\Pi_{n+1} = \{X_0^{n+1}, X_1, ..., X_k\}$ *is an evolution of PN-templates collection* $\Pi_n = \{X_0^n, X_1, ..., X_k\}$, *if* $\exists\ \iota, \phi$, *such that* $X_0^{n+1} = X_0^n \underset{\iota\ \phi}{\oplus} X_i$.

It should be noted that the software implementation of templates collection *evolution* can be made, using a list representation of Petri net elements sets, in no more than $O(N)$ CPU operations, where N - the number of elements in the final net. To do this, at each step of the *evolution*, instead of creating new start template, all changes should be done in current one. Then, in each of the binary merges from $O(N_1+N_2)$ CPU operations only $O(N_2)$ operations, related to copying second net elements and "gluing" places, remain.

Finally, with regard to the program behaviour simulations building, it is possible to formulate the final requirements to PN-templates set.

Definition 13. *Complete PN-templates collection.*
PN-templates collection $\Pi = \{X_i = \langle \Sigma_i, I_i, O_i, M_i \rangle\}$ *is considered to be complete, if:*

1. *All templates have no more than one access point in input interface:* $\forall X_i \in \Pi \to |I_i| \leq 1$.
2. *There is sufficient number of terminator templates:*
   $\forall X_i \in \Pi, \phi \in O_i \to \exists X_j \in \Pi, \iota \in I_j : |O_j| = 0, |\phi| = |\iota|$. [1]

Definition 14. *The resulting template.*
Template $X = \langle \Sigma, I, O, M_0 \rangle$ *is called the resulting temlate, if* $|I| = 0, |O| = 0$.

[1] Theoretically, the existence condition of sufficient number of terminator templates can be refined to reduce the number of templates. So, if there is some set of terminator templates, then can be built a set consisting of all the possible merges of initial terminators with building templates, and the resulting set tested for sufficiency.

With a complete collection of templates as defined in 13, it is possible, using a merge by access points operation 10, to build the resulting template of any predefined size. It is easy to verify that, using start template, each merge operation will result in a new start template that does not have an input interface. Available building templates and access point pairs in interfaces allow to continue build procedure. And merging start template with terminator templates reduces the amount of access points in output interface at the start template until it becames the resulting template.

3. A generation example of Petri net simulating imperative program

Consider as an example the generation of Petri net simulating the behavior of a simple sequential imperative program. The complete collection $\Pi_x = \{X_1..X_{10}\}$ consisting of ten templates is used to build sample net. Let us give drawings of templates and describe each of them in order. The following designations are used in templates representations. Petri net describing the structure of the template is placed in a rectangle. Petri nets are drawn using usual graphical notation in the form of a bipartite directed graph, where places are represented by circles and transitions — by rectangles. Places and transitions are connected by arcs representing input and output incidence functions. At the boundaries of the rectangle, framing template structure, the symbolic images of simple access points by places are drawn in the form of a circles with a sign of the interface it is belonged inside. In this article, all access points of the input interface are placed on the top edge of the rectangle, and all of the output — on the bottom edge. Each place of the access point ordered subset of places is connected by a thin dotted line with the symbol of the access point. Formal descriptions of the template input and output interfaces and its marking are placed inside the rectangle.

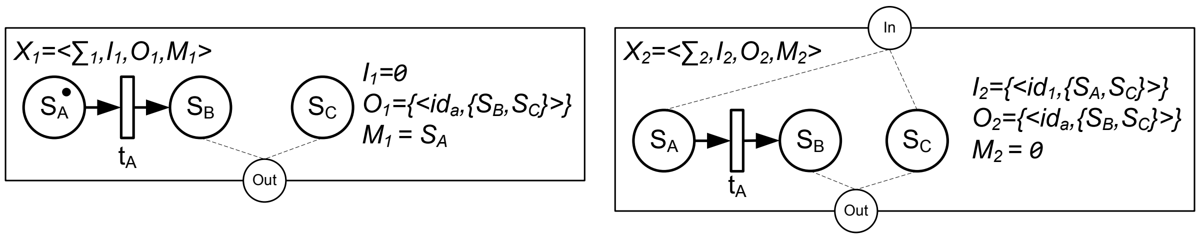
X_1=<$\sum_1$,I_1,O_1,M_1>
S_A
t_A
S_B
S_C
I_1=θ
O_1={<id_a,{S_B,S_C}>}
M_1 = S_A
Out
In
X_2=<$\sum_2$,I_2,O_2,M_2>
S_A
t_A
S_B
S_C
I_2={<id_1,{S_A,S_C}>}
O_2={<id_a,{S_B,S_C}>}
M_2 = θ
Out

Fig 1: Templates of process (left) and linear section (right)

Figure 1 shows first two templates. The first template X_1, called the process template, is modelling begin and end of a sequential process. This is the only start template with a nonzero marking in the described collection, it has no input interface. The initial place with the token is the starting point of the program model, where the program begins its work, and the only transition in template simulates start of the process. Template X_2, called linear section, simulates simple mathematical expression in the imperative program, it differs from start template by absence of marking and presence of input interface.

Next template X_3 is drawn on figure 2 and designed to simulate the behaviour of cycles in the imperative program. First access point of the template input interface is

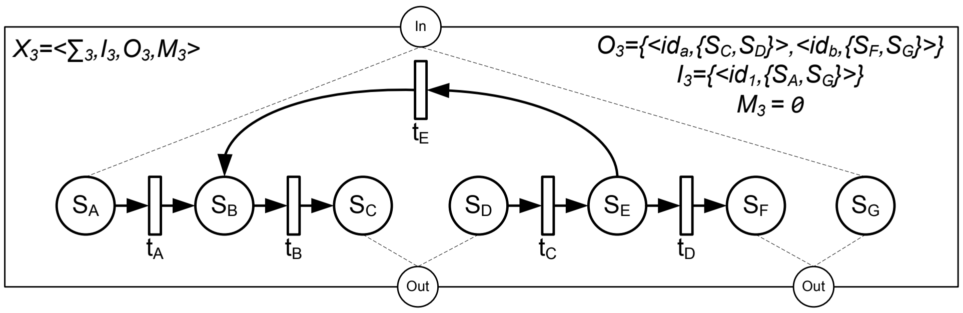

Fig 2: Template of cycle

used to form the cycle body. And second access point - to continue the program after the cycle.

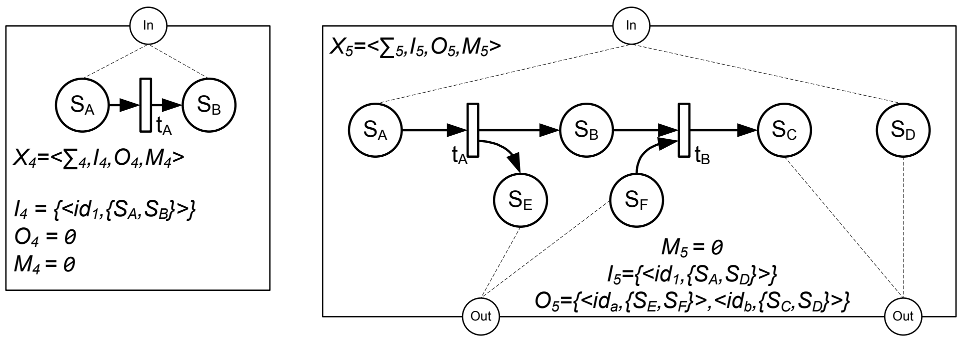

Fig 3: Templates of stub (left) and function call (right) constructions

At the left side of the figure 3 terminator template X_4, called stub, is represented. This template has only one access point of a pair of places in input interface and no access points in output interface, so after the merging with this template the resulting net would have one access point less in output interface. At the right side of the figure there is template X_5, modelling function call in the imperative program. This template has a single access point in input interface and two access points in the output interface. The first access point of the output interface is designed to form the body of the function, the second access point — to continue the program after the function call.

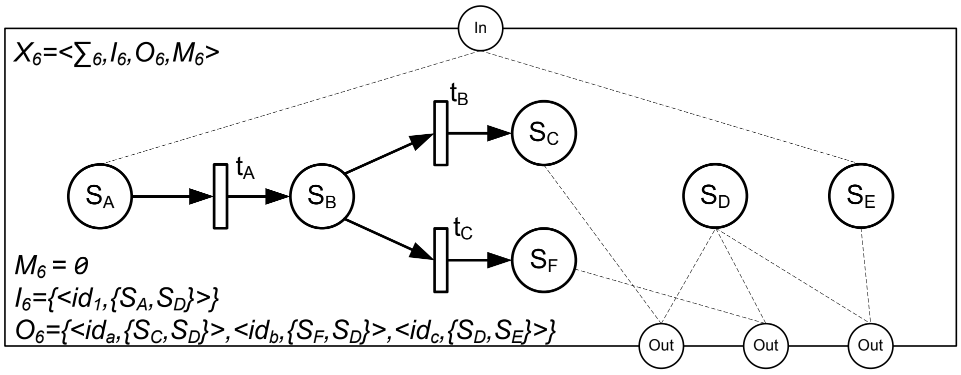

Fig 4: Template of branching operator construction

Figure 4 depicts template X_6 that models an imperative programming language **branching operator** construction. This template has one simple access point in the input interface, consisting of the begin and end places of the template. Three access points in the output interface, each consisting of a pair of places, are designed to simulate the program parts of *then* branch, *else* branch, and to continue the program after the branching operator.

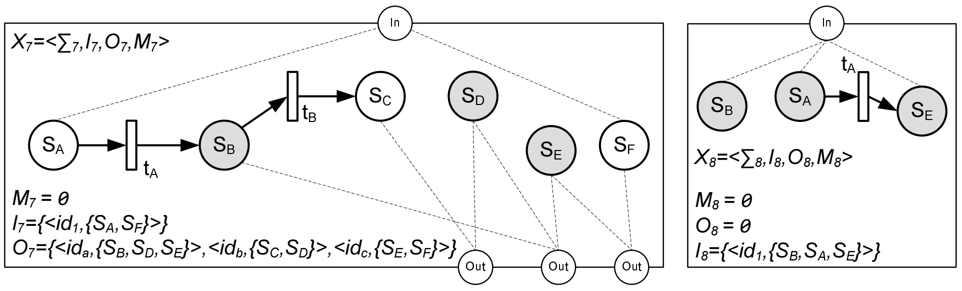
X_7=<∑$_7$,I_7,O_7,M_7>
In
S_A
t_A
S_B
t_B
S_C
S_D
S_E
S_F
M_7 = θ
I_7={<id$_1$,{S_A,S_F}>}
O_7={<id$_a$,{S_B,S_D,S_E}>,<id$_b$,{S_C,S_D}>,<id$_c$,{S_E,S_F}>}
Out
Out
Out
In
t_A
S_B
S_A
S_E
X_8=<∑$_8$,I_8,O_8,M_8>
M_8 = θ
O_8 = θ
I_8={<id$_1$,{S_B,S_A,S_E}>}

Fig 5: The main template of switch (right) and completing as *default* case (right)

Templates X_7, X_8, X_9, X_{10} simulate parts of syntax construction **switch** of imperative programming languages: main template — begin and end of the construction, completing template as *default* case, templates to continue after break operator and continue without break operator accordingly. This templates are shown of figures 5 and 6. Main template X_7 have one access point of a pair of places in input interface and three access points in output interface, designed to continue switch construction, building body of the first execution case of switch construction and to continue program after switch operator. The access point for the continuation of the switch construction has three places, and other two access points — two. Template X_8 has a single access point of the input interface of three places and no output, so it is the terminator for the switch construction, because after the merge operation of the switch construction with template X_8 addition of new cases will be impossible.

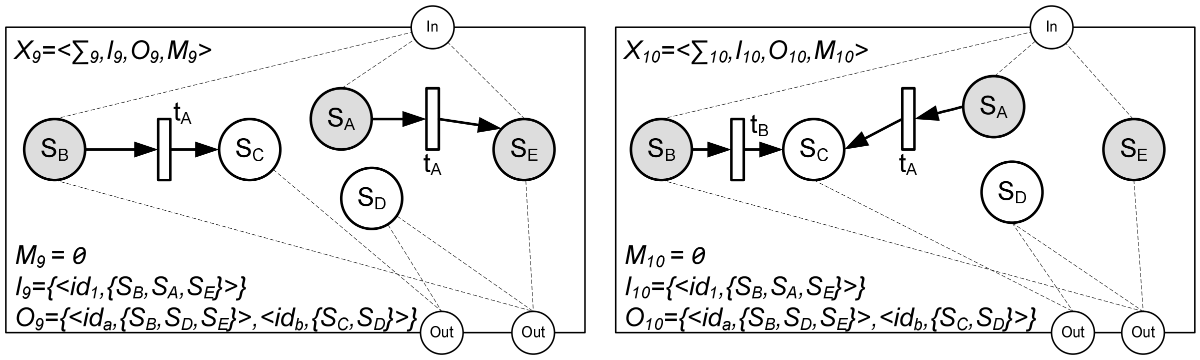
X_9=<∑$_9$,I_9,O_9,M_9>
In
S_B
t_A
S_C
S_A
t_A
S_E
S_D
M_9 = θ
I_9={<id$_1$,{S_B,S_A,S_E}>}
O_9={<id$_a$,{S_B,S_D,S_E}>,<id$_b$,{S_C,S_D}>}
Out
Out
X_{10}=<∑$_{10}$,I_{10},O_{10},M_{10}>
In
S_B
t_B
S_C
t_A
S_A
S_E
S_D
M_{10} = θ
I_{10}={<id$_1$,{S_B,S_A,S_E}>}
O_{10}={<id$_a$,{S_B,S_D,S_E}>,<id$_b$,{S_C,S_D}>}
Out
Out

Fig 6: Templates of switch - continue after *break* operator (left), continue without *break* operator (right)

Figure 6 shows two options X_9, X_{10} of adding a new case to the switch construction. It is due to the single access point of three places in the input interface, this templates can be merged only with a templates from the set of switch construction template. The output interface of these templates has two access points: first access point of three places is designed for the developing of the switch construction, second access point of two places – for constructing the case control flow. Thus, to simulate the behavior of the program in the switch construction it is necessary to use the main switch template, merge it step by step with the required number of cases templates and finish by merging with completing template.

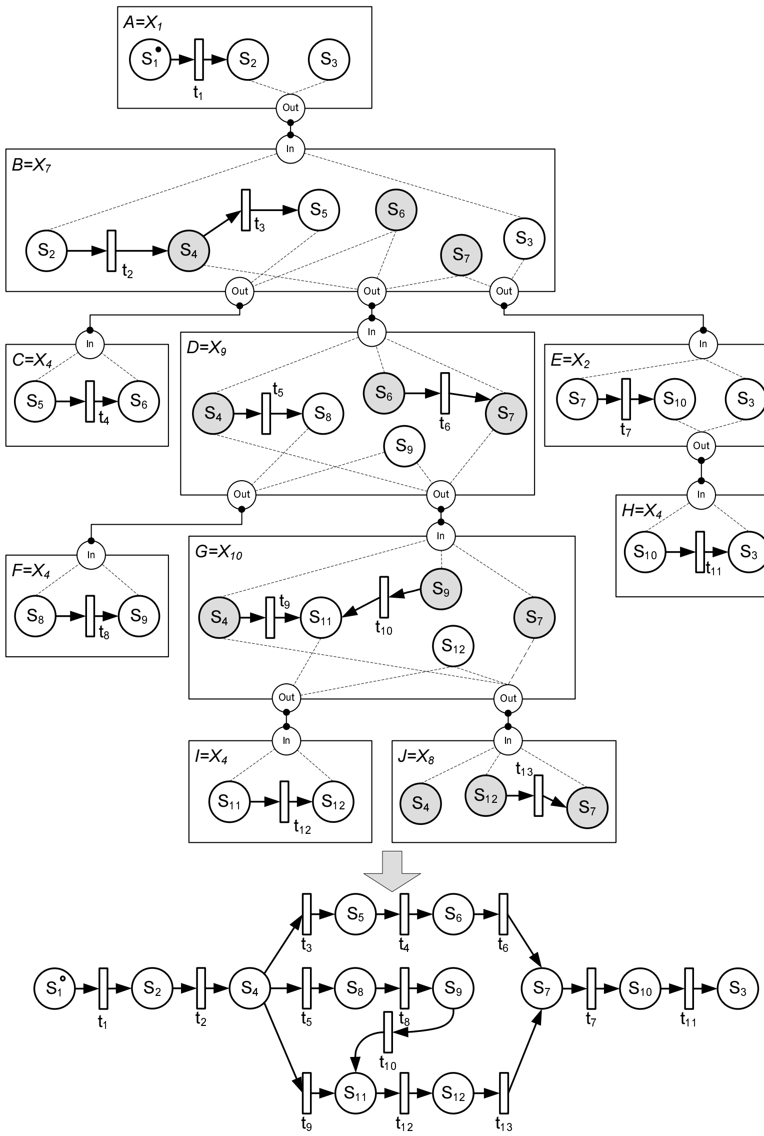

Fig 7: An example of Petri Net construction

Templates collection Π_x described above is a minimal collection for modelling of imperative programs. Let's consider the building process of Petri net simulating imperative program, using this templates collection. For the example of the net building the next templates were used: the process, the switch with three different branches and the stub. Figure 7 shows the diagram of the net construction, with the next used conventions:

- Each template is framed by a rectangle and signed by the ordinal character of the English alphabet and template number. English character indicates the order of the templates merge operations that represents the evolution 12 of a templates collection.

- The lines between simple access points by places show what access points are used in merge operation.

- All access points of the input interface are placed on the upper edge of the rectangle, and of the output — at the bottom. Therefore, the order of templates merging coincides with the rules of reading - from top to bottom, left to right.

- For convenience all the places and transitions of construction templates are renamed to coincide with the resultant places and transitions.

Bottom part of the figure shows the result of templates evolution. This net is similar in behavior to the real program, consisting for the most part of a switch operator, which has three branches: the upper branch with break operator, the middle branch without break operator (without break operator the process continues in next branch) and the default branch.

4. Conclusion

Automatic Petri nets generation is quite often used in the modeling of objects of different application areas, for example, in railway interlocking design [16], large scale biological networks [12], semiconductors manufacturing [8], flexible manufacturing systems [13]. Typically, the description of the object in terms of domain-specific languages is used as the input data for translate procedure, that builds model of the object in terms of Petri nets. The authors proposed a new statement of the problem, when the input data, and the final result are described by Petri nets. Publications conforming that formulation have not yet been met by the authors.

This paper formally describes the method of Petri nets generation on the base of templates, having input and output interfaces in form of sets of simple access points by places, that allows templates merging. A distinctive feature of the method proposed is low computational complexity, since to implement the Petri nets merge operation it is necessary to perform a simple copy of nets elements with the subsequent gluing of beforehand known pairs of places. In the second half of this article a complete templates collection is given and a generation example of Petri net, simulating the behavior of an imperative program. A templates collection used to generate Petri nets, defines the behavior characteristics of the resulting network, so the method proposed has a certain flexibility, which allows to use it not only to build examples of imperative program models, but also to generate other different by behavior Petri nets. In particular, the authors see one of the interesting direction of the proposed method development in its adaptation to generate nets used for verification of analysis tools in “Model Checking Contest @ Petri Net”.

References

[1] Peterson, James Lyle, *Petri Net Theory and the Modeling of Systems*, Prentice Hall, 1981, 290 pp.

[2] Anisimov N.A., Kovalenko A.A., “Towards Petri Net Calculi based on Synchronization via Places”, *Proc. of the 1995 IEEE Symposium on Parallel Algorithms/Architecture Synthesis*, IEEE Press, Japan, 1995, 264–270.

[3] Best, Eike and Devillers, Raymond and Koutny, Maciej, *Petri Net Algebra*, Springer-Verlag New York, Inc., USA, 2001.

[4] International Atomic Energy Agency, A Panel of Experts (2001), *Investigation of an Accidental Exposure of Radiotherapy Patients in Panama/Report of a Team of Experts, 26 May – 1 June, 2001 (PDF)*, Austria: International Atomic Energy Agency, Vienna, 2001.

[5] Anisimov N.A., Golenkov E.A., Kharitonov D.I., “Kompozitsionnal’nyy podkhod k razrabotke parallel’nykh i raspredelennykh sistem na osnove setey Petri”, *Programmirovanie*, 2001, №6, 30–43, (in Russian).

[6] Denaro G. and M. Pezzè, “Petri nets and software engineering”, *Lectures on Concurrency and Petri Nets*, **3098**, Springer-Verlag, 2004, 439–466.

[7] Golenkov E.A., Sokolov A.S., “Metod avtomaticheskogo postroeniya modeli parallel’noy programmy v terminakh setey Petri”, *Vychislitel’nye metody i programmirovanie*, **6**:2 (2005), 77–82, (in Russian).

[8] Mueller Ralph et al., “Automatic Generation of Simulation Models for Semiconductor Manufacturing”, *Proceedings of the 39th Conference on Winter Simulation: 40 Years! The Best is Yet to Come*, WSC ’07, IEEE Press, Piscataway, NJ, USA, 2007, 648–657.

[9] Desel J., “Controlling Petri Net Process Models”, Web Services and Formal Methods, 4th International Workshop, WS-FM (2007, Brisbane, Australia, September 28-29, ed. Marlon Dumas and Reiko Heckel), 2008, 17–30.

[10] Laure Petrucci, “Aggregating views for Petri net model construction”, *In Proc. workshop on Petri Nets and Distributed Systems (PNDS08, associated with Petri Nets 2008)*, 2008, 17–31.

[11] Abdelghani Ghomari, Chaabane Djeraba, “Modelling Multimedia Synchronization using a Time Petri Net Based Approach”, *Advances in Petri Net: Theory and Applications*, eds. Tauseef Aized, Intech, 2010, http://www.intechopen.com/books/advances-in-petri-net-theory-and-applications/modeling-multimedia-synchronization-using-a-time-petri-net-based-approach-.

[12] Chen Ming et al., “Petri net models for the semi-automatic construction of large scale biological networks”, *Natural Computing*, **10**:3 (2011), 1077–1097.

[13] Ballarini P. et al., “Petri Nets Compositional Modeling and Verification of Flexible Manufacturing Systems”, In 7th Annual IEEE Conference on Automation Science and Engineering (CASE 2011), 2011.

[14] Alekseyev Arseniy et al., “Improved Parallel Composition of Labelled Petri Nets”, *ACSD*, eds. Caillaud, Benoît and Carmona, Josep and Hiraishi, Kunihiko, IEEE Computer Society, 2011, 131–140.

[15] Ivan Petko and Stefan Hudák, “General composition for high level Petri nets and its properties”, *Central Europ. J. Computer Science*, **2**:3 (2012), 222–235.

[16] Durmus M.S., Yildirim U., Soylemez M.T., “Automatic Generation of Petri Net Supervisors for Railway Interlocking Design”, *Control Conference (AUCC), 2012 2nd Australian*, IEEE, 2012, 180–185.

[17] Tarasov G.V., Kharitonov D.I, Golenkov E.A.., “Ob odnom predstavlenii funktsii v modeli imperativnoy programmy, zadannoy setyami Petri”, *Modelirovanie i analiz informatsionnykh sistem*, **18**:2 (2011), 18–38, (in Russian).

[18] Kharitonov D., Tarasov G., “Modeling function calls in program control flow in terms of Petri Nets”, *ACSIJ Advances in Computer Science: an International Journal*, **3**:6 12 (November 2014), 82–91.

[19] B. Esther Sunanda, P. Seetharamaiah, "Modeling of Safety-Critical Systems Using Petri Nets", *SIGSOFT Softw. Eng. Notes*, **40**:1 (2015), 1–7.

[20] Kordon F. et al., "Complete Results for the 2015 Edition of the Model Checking Contest", 2015, http://mcc.lip6.fr/2015/results.php.

DOI: 10.18255/1818-1015-2015-4-563-577

Метод генерации примеров моделей программ в терминах сетей Петри

Харитонов Д.И., Голенков Е.А., Тарасов Г.В., Леонтьев Д.В.

получена 1 сентября 2015

В данной работе рассматривается с формальной точки зрения метод построения сетей Петри, имитирующих поведение императивных программ. Примеры сетей Петри с заданными характеристиками являются необходимыми в процессе программирования новых алгоритмов анализа моделей программ, в частности, они могут использоваться для разработки и оптимизации алгоритмов композиции и декомпозиции сетей Петри, построения дерева достижимости, проверки инвариантов и т.д. Способ построения состоит из двух стадий. На первой стадии описываются шаблонные конструкции, из которых будет состоять результирующая сеть, и параметры, с которыми будет выполняться построение. С помощью этих параметров можно регулировать конечный размер, а также абсолютное или относительное количество определённых конструкций в результирующей сети. На второй стадии с помощью автоматического итерационного процесса может быть сгенерирована сеть Петри любого размера, ограниченного оперативной памятью компьютера. В первом разделе статьи приводится необходимый минимум определений и вводится новый вариант операции композиции сетей Петри по местам. Свойства коммутативности и ассоциативности бинарного вида предложенной операции позволяют сливать несколько сетей Петри в произвольном порядке. Далее вводится понятие шаблонной конструкции в виде маркированной сети Петри, обладающей входным и выходным интерфейсами, а также правилами композиции шаблонных конструкций с использованием этих интерфейсов. Множество шаблонных конструкций объединяются в набор, для которого определяются правила эволюции. Свойство полноты набора гарантирует, что в результате эволюции набора будет получена сеть Петри, имитирующая поведение императивной программы. В статье приводится вариант полного набора шаблонных конструкций и пример генерации сети Петри, имитирующей последовательную императивную программу.

Статья публикуется в авторской редакции.

Ключевые слова: модель программы, модель потока управления, теория сетей Петри, объект сети Петри

Для цитирования: Харитонов Д.И., Голенков Е.А., Тарасов Г.В., Леонтьев Д.В., "Метод генерации примеров моделей программ в терминах сетей Петри", *Моделирование и анализ информационных систем*, **22**:4 (2015), 563–577.

Об авторах:
Харитонов Дмитрий Иванович, orcid.org/0000-0003-3359-2383, канд. техн. наук, ст. науч. сотр.,
Федеральное государственное бюджетное учреждение науки Институт автоматики и процессов управления Дальневосточного отделения РАН, ул. Радио, д. 5, г. Владивосток, Россия, 690041, e-mail: demiurg@dvo.ru

Голенков Евгений Александрович, orcid.org/0000-0002-8148-3504, канд. физ.-мат. наук, ст. науч. сотр.,
Федеральное государственное бюджетное учреждение науки Институт автоматики и процессов управления Дальневосточного отделения РАН, ул. Радио, д. 5, г. Владивосток, Россия, 690041, e-mail: golenkov@dvo.ru

Тарасов Георгий Витальевич, orcid.org/0000-0001-8855-7388, науч. сотр.
Федеральное государственное бюджетное учреждение науки Институт автоматики и процессов управления Дальневосточного отделения РАН, ул. Радио, д. 5, г. Владивосток, Россия, 690041,
Дальневосточный Федеральный Университет, ул. Суханова, д. 8, г. Владивосток, Россия, 690950, e-mail: george@dvo.ru

Леонтьев Денис Валерьевич, orcid.org/0000-0002-5116-3008, аспирант,
Федеральное государственное бюджетное учреждение науки Институт автоматики и процессов управления Дальневосточного отделения РАН, ул. Радио, д. 5, г. Владивосток, Россия, 690041
Дальневосточный Федеральный Университет, ул. Суханова, д. 8, г. Владивосток, Россия, 690950, e-mail: devozh@dvo.ru

Благодарности:
Работа выполнена при финансовой поддержке программы фундаментальных исследований РАН по приоритетным направлениям "Фундаментальные проблемы математического моделирования", проект № 0262-2014-0157.