Universal Inhibitor Petri Net

Dmitry Zaitsev

International Humanitarian University
Department of Information Technology
Fontanskaya Doroga st., 33, Odessa 65009, Ukraine
http://member.acm.org/~daze

Abstract. The universal inhibitor Petri net was constructed that executes an arbitrary given inhibitor Petri net. The inhibitor Petri net graph, its marking and the transitions firing sequence were encoded as 10 scalar nonnegative integer numbers and represented by corresponding places of universal net. The algorithm of inhibitor net executing that uses scalar variables only was constructed on its state equation and encoded by universal inhibitor Petri net. Subnets which implement arithmetic, comparison and copying operations were employed.

Keywords: universal inhibitor Petri net, universal Turing machine, encoding, algorithm

1 Introduction

It is known, that inhibitor, synchronous, priority and other extended Petri net classes constitute a universal algorithmic system [1,2]. For such universal algorithmic systems as Turing machines, there are known examples of universal Turing machine construction [3]. In this connection it is of a definite interest the construction of a universal Petri net which executes an arbitrary given Petri net that is the goal of the present paper.

2 The Concept of a Universal Petri Net

The universal net is constructed in the class of inhibitor Petri nets [1,2], the corresponding universal inhibitor Petri net is denoted as UIPN. Considering nondeterministic character of Petri net dynamics the most close analog is nondeterministic Turing machine [3].

As it is of interest the constructing of the universal Petri net with a fixed structure, the only way of input and output information representation is the marking of a fixed number of definite UIPN places. Therefore, it is necessary to give rules of a biunique encoding of Petri net graph and its marking by a fixed quantity of nonnegative integer numbers. Let there are given the corresponding encoding rules and sXIPN is the code of Petri net XIPN graph and sQXIPN is the code of the marking QXIPN.

The concept of reachable marking in Petri net implies the existence of the corresponding enabled sequence of transitions firing [1,2]. But the usage of only marking QXIPN in the definition of UIPN does not guarantee the obtaining of all the enabled sequences of transitions firing of the net XIPN. Let definite rules of the transitions firing sequences encoding are given and sZQXIPN is a code of the enabled transitions firing sequence ZQXIPN which moves Petri net XIPN from marking Q0XIPN to marking QXIPN. Then the functioning of UIPN can be represented as the scheme shown in fig. 1.

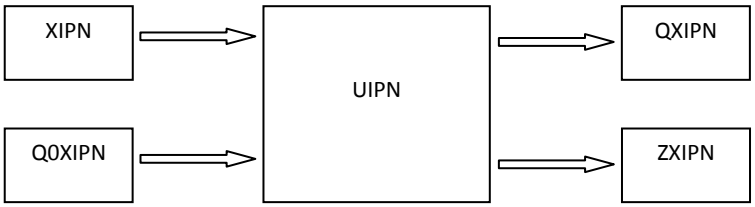


Fig. 1. The scheme of universal inhibitor Petri net UIPN functioning.

Definition 1. Petri net UIPN is a universal inhibitor Petri net if and only if for an arbitrary given inhibitor Petri net XIPN and its initial marking Q0XIPN the net UIPN stops in the marking (sQXIPN,sZQXIPN), where marking QXIPN is reachable in XIPN with the transitions firing sequence ZQXIPN and any marking (sQXIPN,sZQXIPN) which UIPN stops in is a code of a marking QXIPN reachable in XIPN from initial marking Q0XIPN with the transition firing sequence ZQXIPN.

The requirement of the UIPN stopping possibility even in case of a nondead marking QXIPN of the net XIPN is connected with the provisioning the checkpoint (observance) of any reachable marking (and transitions firing sequence) and abstracting from the implementation of UIPN; otherwise it is necessary to add some extra restrictions for the exclusion out of the observance the intermediate markings of UIPN.

3 Formal Representation of Inhibitor Petri Net

Graph of inhibitor Petri net [1,2] is a four-tuple $G = (P, T, B, D)$, where $P = \{p_1, .., p_m\}$ is a finite number of nodes named places, $T = \{t_1, .., t_n\}$ is a finite number of nodes named transitions and the mappings $B: P \times T \to \mathbb{N} \cup \{-1\}$ and $D: T \times P \to \mathbb{N}$ define the input and output arcs of transitions correspondingly together with their multiplicity, $\mathbb{N}$ is the set of nonnegative integer numbers; zero value of mappings B, D denote the absence of the arc, nonzero – the arc multiplicity, the special value -1 denotes the inhibitor arc. The mappings can be represented by the corresponding matrices: $B = \|b_{i,j}\|, b_{i,j} = B(p_j, t_i)$ и $B = \|d_{i,j}\|, d_{i,j} = D(t_i, p_j)$.

The state of net is named a marking and represented by the mapping $Q: P \to \mathbb{N}$, that gives the number of dynamic elements – tokens within places of net. Inhibitor Petri net [1,2] is a couple $N = (G, Q_0)$, where G is the net graph and Q_0 – its initial marking. The marking can be represented by the corresponding vector: $Q =$

$\|q_j\|, q_j = Q(p_j)$. Thus, the inhibitor Petri net is given by the pair of numbers, pair of matrices and a vector: N=(m, n, B, D, Q_0).

The dynamics of inhibitor net constitutes a step-by-step process of its marking transformation as a result of transitions firing [1,2] and can be formally represented by the following system:

$$\begin{cases} q_j^k = q_j^{k-1} - x(b_{l,j}) + d_{l,j}, j = \overline{1,m} \\ u(t_i) = \bigwedge\limits_{j=1,m} ((y(b_{i,j}) \wedge (q_j^{k-1} = 0)) \vee (\bar{y}(b_{i,j}) \wedge (q_j^{k-1} \geq b_{i,j})), i = \overline{1,n} \\ u(t_l) = 1, l \in \overline{1,n} \\ k = 1,2,\ldots \\ x(b) = \begin{cases} b, b \geq 0 \\ 0, b = -1 \end{cases}, y(b) = \begin{cases} 0, b \geq 0 \\ 1, b = -1 \end{cases}. \end{cases} \quad (1)$$

The first line of the system (1) describes the marking transformation at the transition t_l firing; the function $u(t_i)$ in the second line defines the transition t_i enabling condition at the current step k, the third line defines nondeterministic choice of the firing transition t_l out of the set of enabled transitions, the fourth line gives the order of steps sequence; auxiliary mappings x and y serve for defining the marking decrement and inhibitor arc recognition respectively.

4 Encoding of Inhibitor Petri Net

In the present section a representation of encoding of inhibitor Petri net, its current marking and corresponding transitions firing sequence is obtained in the form of the marking of 10 special places of universal net UIPN (fig. 2). The examples of nets encoding are shown in Appendix A.

4.1 Encoding of a Vector

Let Q is a vector (line), containing m nonnegative integer elements; suppose that elements indexing is started from zero. Let also the following value is calculated

$$r = \max_j q_j + 1.$$

The vector encoding function is defined as

$$s = \varphi(Q) = \sum_{j=0}^{m-1} r^j \cdot q_j.$$

Statement 1. The vector encoding function is injective.

The corresponding decoding function is represented as

$$q_i = \psi(s, j) = (s \; div \; r^j) \; mod \; r.$$

Inherently, the defined encoding is the form of numbers representation in the radix notation with the radix r.

The encoding can be implemented recursively

$$s_j = s_{j-1} \cdot r + q_{m-1-j}, s_0 = q_{m-1}, j = \overline{1, m-1}, \tag{2}$$

where the code of the vector Q equals to $s = s_{m-1}$.

The decoding can be implemented recursively also

$$q_j = s_{m-1-j} \bmod r, s_{m-1-(j+1)} = s_{m-1-j} \ div \ r, s_{m-1} = s, \quad j = \overline{0, m-1}. \tag{3}$$

4.2 Encoding of a Matrix

Let A is a $n \times m$ matrix with nonnegative integer values of elements; suppose that elements indexing is started from zero. Let also the following value is calculated

$$r = \max_{i,j} a_{i,j} + 1.$$

While encoding, let us represent the matrix as a vector with the expansion on lines. Then the matrix A is encoded as

$$s = \varphi(A) = \sum_{i=0}^{n} \sum_{j=0}^{m} r^{(m \cdot i + j)} \cdot a_{i,j.}$$

Statement 1. The matrix encoding function is injective.

The corresponding decoding function is represented as

$$a_{i,j} = \psi(A, i, j) = \left(s \ div \ r^{(m \cdot i + j)}\right) mod \ r.$$

The encoding can be implemented recursively

$$s_v = s_{v-1} \cdot r + a_{i,j}, s_0 = a_{n-1,m-1}, \quad v = \overline{1, n \cdot m - 1}, i = v \ div \ n, j = v \bmod n, \tag{4}$$

where the code of the matrix A equals to $s = s_{n \cdot m-1}$.

The decoding can be implemented recursively also

$$a_{i,j} = s_{n \cdot m-1-v} \bmod r, s_{n \cdot m-1-(v+1)} = s_{n \cdot m-1-v} \ div \ r, s_{n \cdot m-1} = s, \quad v = \overline{0, n \cdot m - 1}, i = v \ div \ n, j = v \bmod n. \tag{5}$$

4.3 Encoding of Inhibitor Petri Net Graph

The graph is represented by the pair of matrices B and D. Usually the zero value of the matrix element indicates the absence of the corresponding arc, nonzero – its multiplicity. The representation of inhibitor arcs of matrix B require supplementary agreements to avoid negative values. Let k is the multiplicity of arc, then for its representation the value $k + 1$ is used; the value of 1 is reserved for the inhibitor arc representation.

It is reasonable the separate encoding according to (4) and storing in separate places the codes of matrices B and D, as well as the corresponding values of r. For the storing of the encoded Petri net graph, 6 corresponding places with names m, n, sB, rB, sD, rD are used shown in fig. 2 which marking contains the values m, n, sB, rB, sD, rD respectively.

4.4 Encoding of Marking

The marking of a Petri net containing m places is given by the vector Q of size m with the nonnegative integer components $q_i = Q(p_i)$. For the storing of the marking encoded according to (2), 3 places with the names m, sQ, rQ are used shown in fig. 2 which marking contains values m, sQ, rQ respectively.

4.5 Encoding of the Transitions Firing Sequence

The transitions firing sequence Z of length k is represented by the vector $\bar{Z}$ of size k with nonnegative integer components $z_j = i$, where i is the number of transition t_i firing on the step j. For the storing of the encoded according to (2) sequence, 3 places with the names k, sZ, n are used shown in fig. 2 which marking contains values k, sZ, n respectively.

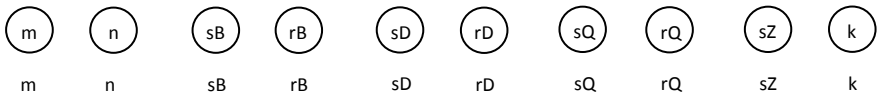


Fig. 2. The representation of the Petri net and transitions firing sequence encoding.

Note that places m, n are used as the parameters for the encoding (decoding) the Petri net graph, marking and transitions firing sequence.

4.6 Encoding of the Enabled Transitions Set

The enabled transitions set of Petri net is auxiliary information for the nondeterministic choice of the firing transition t_l $(u_l = 1)$ on the current step. For the representation of the enabled transitions set, the vector $\bar{u}$ of size n is used which components are the enabling indicators u_i of the corresponding transitions $t_i, i =$

$\overline{0, n-1}$, calculated according to (1). Then for the encoding of $\bar{u}$, the rules of the vector encoding (2) are applied at $r = 2$.

5 Algorithm of Inhibitor Petri Net Executing

On the system (1) according to the chosen way of the encoding of Petri net graph, marking and transitions firing sequence let us construct the algorithm AUIPN of inhibitor Petri net executing using C-like pseudo language:

```
void AUIPN()
{
        uint u, l;

        inputXIPN();
        k=1; sZ=0;
        while(NonDeterministic())
        {
                CheckFire(&u);
                if(u==0) goto out;
                PickFire(u, &l);
                Fire(l);
                mul_add(&sZ,n,l-1);
                k++;
        }
        out:    outputXIPN();
}
```

The following variables are used: u – the code of enabled transitions indicator, l – the number of the firing transition, k – the number of the current step; procedures: CheckFire – checking the transitions enabling conditions, PickFire – the firing transition choice, Fire – the firing of the transition; NonDeterministic – nondeterministic choice of a number belonging to the set $\{0,1\}$. The algorithms of the auxiliary procedures mod_div, mul_add are the following:

```
void mod_div(&m,&x,y)
{
        (*m) = (*x) mod y;
        (*x) = (*x) div y;
}

void mul_add(&x,y,z)
{
        (*x) = (*x) * y + z;
}
```

The algorithm of the procedure CheckFire is the following:

```
void CheckFire(uint *u)
{
        uint i, j, qj, bij, ui, uij;
        uint sB1, sQ1;

        sB1=sB; &u=0;
        for(i=n; i>0; i--)
        {
```

```
                sQ1=sQ;
                ui=1;
                for(j=m; j>0; j--)
                {
                        mod_div(&qj,&sQ1,rQ);
                        mod_div(&bij,&sB1,rB);
                        uij=1;
                        if(bij==0) continue;
                        bij--;
                        if(bij==0) uij=(qj==0);
                        else uij=(qj>=bij);
                        ui=ui && uij;
                }
                mul_add(&u,2,ui);
        }
}
```

Lemma 1. Algorithm CheckFire creates the set of transitions enabled in the current marking.

Proof. The algorithm constitutes the sequential computation of the vector $\bar{u}$ components according to the second line of the system (1) and their simultaneous encoding (2) into the variable u after the calculation of the current component in the variable ui. The loop on the variable i defines the exhaustion of all the transitions, the nested loop on the variable j defines the exhaustion of all the places for the chosen transition. The order of the sequential decoding of matrix B and vector Q elements corresponds to the order of the loops indices modification according to (3) and (5). □

The algorithm of the procedure PickFire is the following:

```
void PickFire(uint u, uint *l)
{
        uint ui, i;

        i=0;
        while(u>0)
        {
                mod_div(&ui,&u,2);
                i++;
                if(ui==0) continue;
                if(NonDeterministic()) goto out;
        }
out:    *l=i;
}
```

Lemma 2. Algorithm PickFire executes the choice of an arbitrary firing transition from the set of enabled transitions.

Proof. The condition of the firing transition choice corresponds to the third line of the system (1) as well as to the order of the vector $\bar{u}$ decoding according to (3). For the nondeterministic choice of the firing transition the function NonDeterministic is used for the exit out of the loop. The condition $u > 0$ provides the loop completion after the last enabled transition processing which is chosen as the firing at least. □

The algorithm of the procedure Fire is the following:

```
void Fire(uint l)
```

```
{
        uint rQ1, maxQ1, shift, qj, bij, dij, j;
        uint sB1, sD1, sQ1;

        sB1=sB; sD1=sD; sQ1=0; rQ1=rQ+rD-1; maxQ1=0;

        shift=(n-l)*m;
        while(shift--)
        {
                mod_div(&b,&sB1,rB);
                mod_div(&d,&sD1,rB);
        }

        for(j=m; j>0; j--)
        {
                mod_div(&qj,&sQ, rQ);
                mod_div(&bij,&sB1, rB);
                if(bij>0) bij--;
                dij=mod_div(&sD1, rD);
                qj=qj-bij+dij;
                maxQ1=max(qj,maxQ1);
                mul_add(&sQ1,rQ1,qj);
        }
        sQ=0; rQ=maxQ1+1;

        for(j=m; j>0; j--)
        {
                mod_div(&qj,&sQ1,rQ1);
                mul_add(&sQ,rQ,qj);
        }
}
```

Lemma 3. Algorithm Fire implements the marking transformation as a result of the specified transition firing.

Proof. The algorithm implements the recalculating of the marking according to the first line of the system (1) and the described way of the matrices B, D and the vector Q decoding according to (5) and (3). The value of the variable shift corresponds to the number of the passing through elements for the positioning to the beginning of the firing transition line with the number l. Then into the first loop on the variable j the preliminary recalculating of the marking code (2) is executed into the variable sQ1; at that the value of rQ1 is used which provides the storing of the maximal possible value of the new marking element rQ+rD-2. For the avoiding the rQ growth, into the second loop on the variable j the final recalculating of the marking code (2) is executed into the variable sQ according to the actual value of the maximal element maxQ1. □

Theorem 1. Algorithm AUIPN implements the dynamics of an arbitrary given inhibitor Petri net.

Proof. Let us show that the algorithm AUIPN recalculates the marking of inhibitor Petri net according to the system (1) and stores the employed transitions firing sequence. The algorithm of the step executing is represented by the loop while of AUIPN and completely corresponds to the system (1). At the beginning, the procedure CheckFire determines the enabled transitions set and forms the code (2) of the corresponding enabled transitions indicator u (Lemma 1). At the absence of the

enabled transitions $u == 0$, the algorithm stops that corresponds to a dead marking. The procedure PickFire implements nondeterministic choice of the firing transition from the set of the enabled transitions; the variable l returns the firing transition number (Lemma 2). The procedure Fire implements the current marking transformation as a result of the transition with the number l firing and its simultaneous encoding (2) (Lemma 3). Then into the code (2) of the transitions firing sequence sZ is added the number l and the value of the current step k is incremented by unit. Nondeterministic exit out of the loop corresponds to the Definition 1. □

Algorithm AUIPN was also encoded in C language using the library MPI for the representation of lengthy integers and tested on a series of Petri nets.

Theorem 2. Algorithm AUIPN can be represented by an inhibitor Petri net.

The Theorem 2 proof is the immediate consequence of the facts that inhibitor Petri net is a universal algorithmic system [1] and the algorithm AUIPN uses nonnegative integer scalar variables only which values can be represented by the marking of the corresponding Petri net places.

For the constructive proof of Theorem 2, the corresponding net is constructed on the algorithm AUIPN in the following sections of the work.

6 Principles of Algorithms Encoding by Inhibitor Petri Net

There are known various approaches to the algorithm encoding by a Petri net based on the principles of combining data flows and control flows [2,4,5]. Let us employ the direct encoding of the basic C language operators for the representing of single control flow. Each of variables is represented by the corresponding place of Petri net; all the variables are static global (fig. 3). The control flow is modeled by the trace of a single token passage from initial place start to the final place finish.

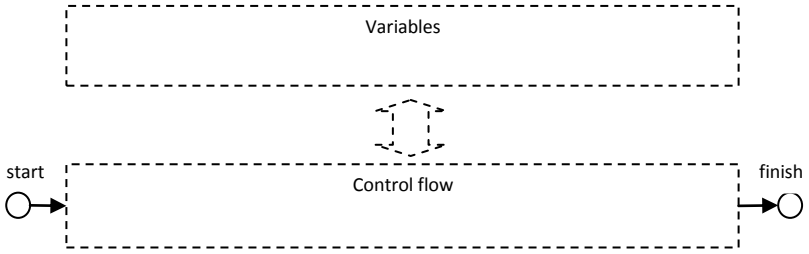


Fig. 3. Overall organization of the net UIPN.

For the unified organization of work with variables let us represent the operators of the programming language in the form shown in fig. 4.

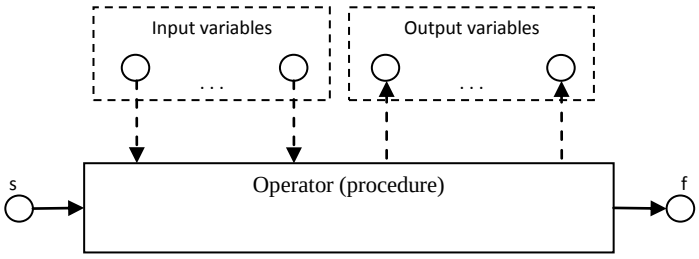


Fig. 4. Representation of the programming language operator.

To provide the reentering the control flow through the operators (procedures) let us adopt the following agreements: all the internal places have zero marking; before the beginning of the work the input variables are copied into the input places of the operator; the work of the operator is launched by a token put into the place start (s); the operator finishes its work at the hitting the place finish (f) by the token; at the completion of work all the places of the operator are empty excepting the output places which contain the result. Dashed arcs denote the following extra rules of the forming the values of the operator input and output variables: at the launch the content of the variable is copied into local input place of the operator; after the completion the variable is cleaned and the value from the local output place of the operator is moved into it (fig. 5).

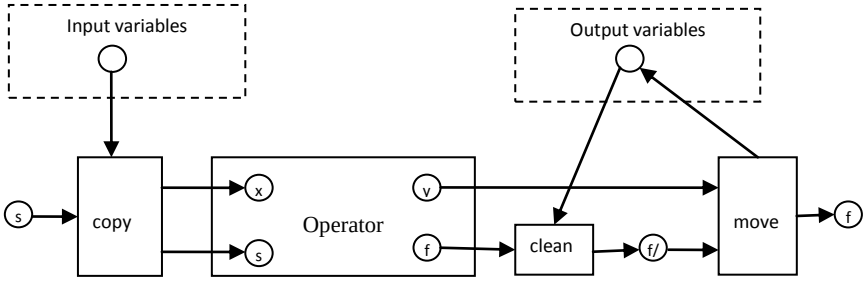


Fig. 5. The forming of input and output variables.

In case of a few variables the chains of copy are created for the sequential copying of input variables and the chains of clean, move for the moving of the output variables values. The sequence of clean, move is denoted as assign. The represented scheme provides the correct work with variables in general case. In some cases the work with variables can be optimized, when they are temporary or input and output at the same time. For the expressions calculating the approach of data flows [2] can be implemented: the executing of operations is ordered according to their priorities; input places of operations are fused with output places of the next operation.

Let us consider the basic control constructions of the programming language: sequence, conditional (unconditional) branch, loop. Let us abstract from the used variables.

Lemma 4. Algorithmic control constructions of the programming language can be encoded by inhibitor Petri net in the following way (fig. 6):

Name	Form	Net
Sequence	`operator1;` `operator2;`	a)
Branch	`if(condition()) then operator1; else operator2;`	b)
Loop while	`while(condition()) operator;`	c)
Loop for	`for(i=n;i>0;i--) operator;`	d)

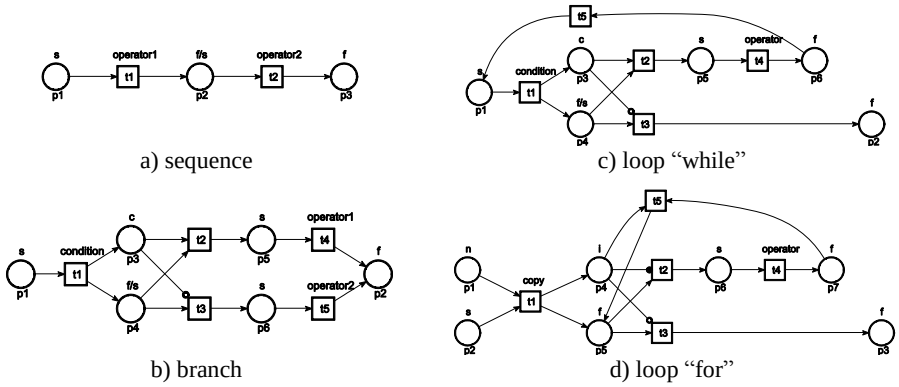


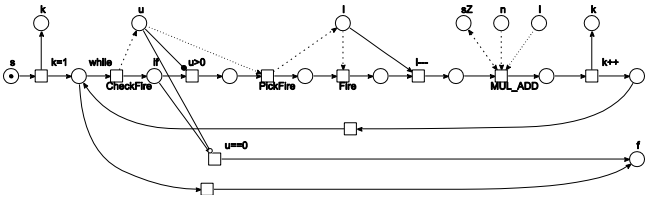
Fig. 6. Encoding of the programming language control constructions.

For each control construction the correctness of its representation can be proven by the way of classifying all the enabled transitions firing sequences and their comparison with the order of operators execution into the constructions of the programming language [2]. Note that according to fig. 6a) the operators superposition at the program encoding is implemented by the merging (fusion) of the output place f of the first operator with the input place s of the second operator.

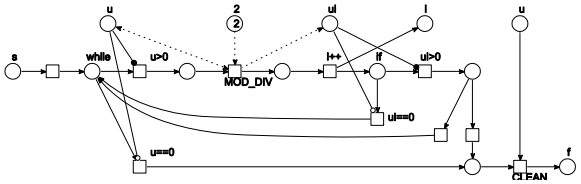
There are known the representations of basic algebraic and logic operations by Petri nets [2,6]. In some cases it is convenient the direct representation of the most used actions such as, for example, mod_div and mul_add for the decoding and encoding of Petri nets. In Appendix B the nets implementing the operations used in the algorithm AUIPN are listed. For the graphical representation of inhibitor arc the hollow circle at the end of arc is used. Arc with the filled circle at its end denotes the couple of arcs with the opposite direction and equal multiplicity; they are used for the checking of a place marking.

Lemma 5. Nets listed in Appendix B implement the specified operations.

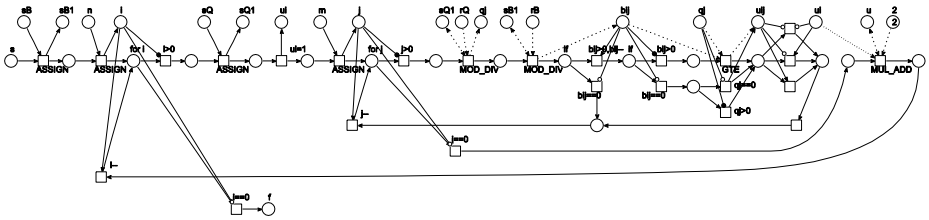
For each of the represented nets it is possible to bring the proof of the correct implementation of the specified operation on the base of all the enabled transitions firing sequences classification [2,6].



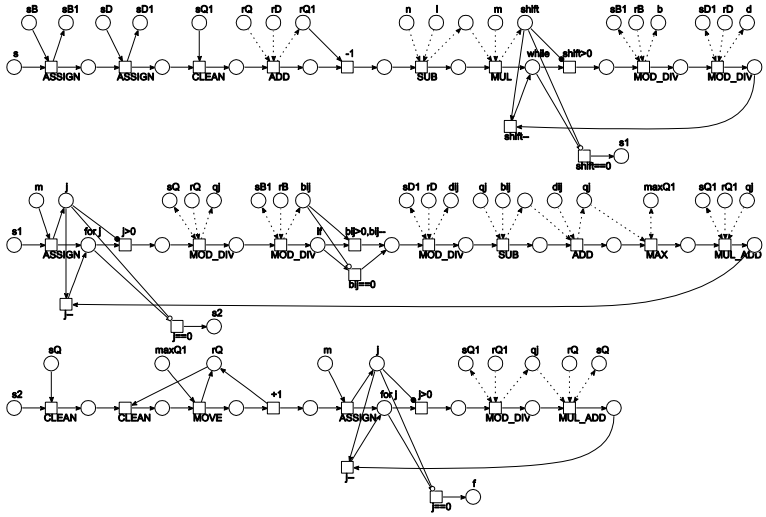
a) UIPN



b) PickFire



c) CheckFire



d) Fire

Fig. 7. Universal inhibitor Petri net UIPN.

7 Composing Universal Inhibitor Petri Net UIPN

Let us encode the algorithm AUIPN of universal inhibitor Petri net work by inhibitor Petri net according to the rules described in Section 6. Note that Lemma 4 and Lemma 5 lists all the control constructions and all the operations employed in the algorithm AUIPN. The net UIPN represented in fig. 7 is obtained. For the representing of the algorithm variables, fused places are used: all the places with the same name are logically the same place; fused places simplifies the graphical representation of the net. Let us suppose that before the net UIPN launch, the code of target (executing) net XIPN is loaded into places shown in fig. 2 and after the stopping of the net UIPN, the code of the marking and the transitions firing sequence of the net XIPN is read from the corresponding places.

Dashed arcs denotes considered in Section 6 agreements on the input and output variables copying. Bidirectional arcs are used for the work with variables which are the both input and output; in this case the copying can be optimized applying twice move without cleaning. In some cases for the copying of an input variable together with its cleaning it is reasonable the usage of move instead of copy; as the corresponding notation the dotted arc is used. The substitution of a transition implies the copying of the corresponding subnet with the merging (fusion) of contact places. In general case the transition substitution requires the indication of input and output places mapping; in the listed nets the places mapping is defined implicitly by the context of the used operations and is not indicated.

Theorem 3. Net UIPN is the universal inhibitor Petri net.

The Theorem 3 proof directly follows from Theorem 1 and the correctness of used rules of sequential algorithm encoding by inhibitor Petri net (Lemma 4) and the correctness of nets implementing the used operations (Lemma 5).

Note that net UIPN is represented in a component-wise way according to the used procedures, operations and the rules of work with variables. There is of a definite interest the binding of UIPN in the form of united inhibitor Petri net and its execution in the environment of a simulating system that simulates the firing of transitions.

8 Conclusions

In the present work the universal inhibitor Petri net was constructed that executes an arbitrary given inhibitor Petri net.

It is possible the constructing of universal nets in other classes of Petri nets which are the universal algorithmic system [2]: priority, synchronous, timed. Moreover, it is possible the combined constructing, for example, of inhibitor net that executes an arbitrary synchronous net.

There are known examples of universal Turing machines constructing with the minimal number of used symbols/states [7,8]. In this connection there is of a definite interest the constructing of universal Petri net with the minimal number of places (transitions), the minimal value of the marking.

References

1. Agerwala T. A Complete Model for Representing the Coordination of Asynchronous Processes, Baltimore, John Hopkins University, Res. Rep. No. 32, July 1974.
2. Kotov V. Petri Nets, Moscow, Nauka, 1984, 160 p. In Russ.
3. The Universal Turing Machine. A Half-Century Survey / Rolf Herken (ed.), Springer-Verlag, Wien New York, 1994, 609 p.
4. Best E, Devillers R., Koutny M. Petri Nets, Process Algebra and Concurrent Programming Languages. Lecture Notes in Computer Science, Vol. 1492: Lectures on Petri Nets II: Applications / Reisig, W.; Rozenberg, G. (eds.), 1998.
5. Goltz U. On Representing CCS Programs by Finite Petri Nets. Proc. MFCS-88, Springer-Verlag Lecture Notes in Computer Science Vol.324, 339-350 (1988).
6. Sleptsov A.I. State equation and equivalent transformations of loaded Petri nets (algebraic approach) // Formal models of parallel computations: Proceedings of all-USSR conference, Novosibirsk (Russia), 1988, p. 151-158. In Russ.
7. Minsky M. Size and structure of universal Turing machines using tag systems. In Recursive Function Theory, Provelence, 1962. AMS, vol. 5, p. 229-238.
8. Rogozhin Y. Small universal Turing machines. TCS, 168(2):215-240, 1996.

Appendix A: Examples of Nets Encoding

1) Petri net graph

Net	m	n	sB	rB	sD	rD
add	6	4	21180169496	3	282946	2
max	8	8	254813592433189871074065241412	3	293862152152879368	2
mul	10	9	646549072061101455668889034663481743952654	3	1935225908529245455975681	2

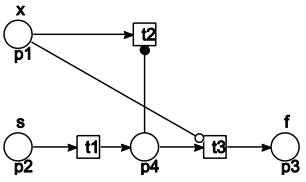
2) Marking

Net	Marking	Q	sQ	rQ
add	addQ0	(2,3,1,0,0,0)	2880	4
add	addQ	(0,0,0,5,1,0)	186	6
max	maxQ0	(2,3,1,0,0,0,0,0)	46080	4
max	maxQ	(0,0,0,3,1,0,0,0)	832	4
mul	mulQ0	(2,3,1,0,0,0,0,0,0,0)	737280	4
mul	mulQ	(0,0,0,6,1,0,0,0,0,0)	722701	7

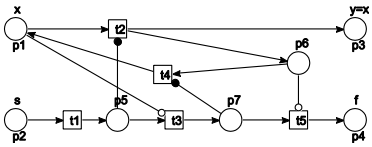
3) Transitions firing sequence

Net	Q0	Q	Z	sZ	k
add	addQ0	addQ	t1,t3,t2,t2,t3,t3,t4	2411	7
max	maxQ0	maxQ	t1,t2,t2,t6,t7,t8	4983	6
mul	mulQ0	mulQ	t1,t2,t4,t4,t5,t6,t6,t7,t2,t4,t4,t5,t6,t6,t7,t2,t4,t4,t5,t6,t6,t7,t3,t9,t9,t8	109815712212339723705298	26

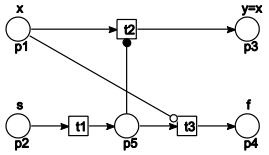
Appendix B: Implementation of Used Operations



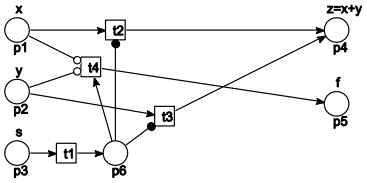
CLEAN ($x = 0$)



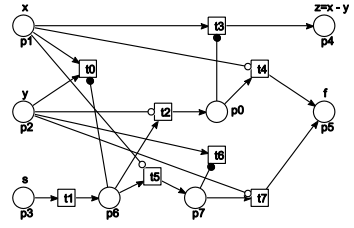
COPY ($y = x$)



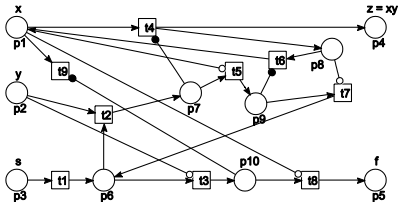
MOVE ($y = x, x = 0$)



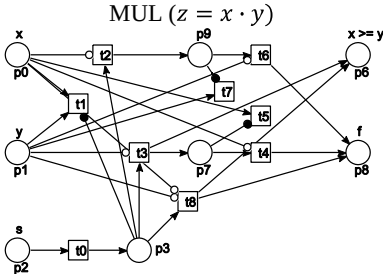
ADD ($z = x + y$)



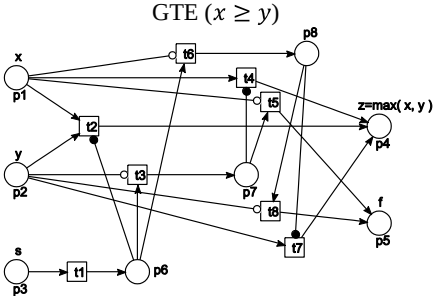
SUB ($z = x \dot{-} y$)



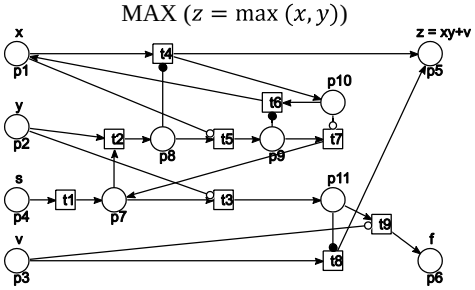
MUL ($z = x \cdot y$)



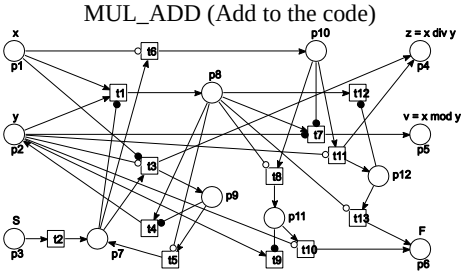
GTE ($x \geq y$)



MAX ($z = \max(x, y)$)



MUL_ADD (Add to the code)



MOD_DIV (Extract from the code)