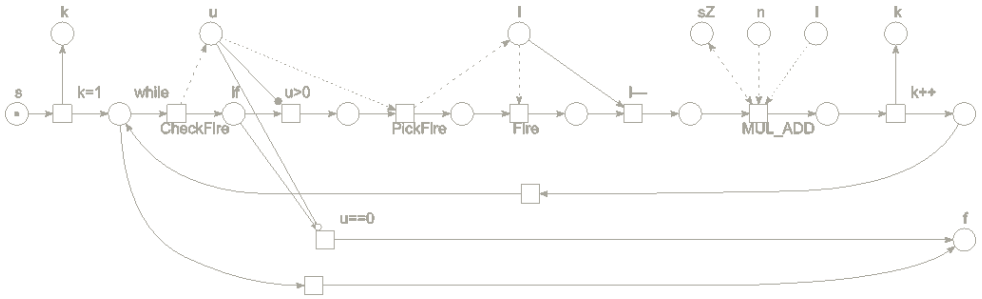
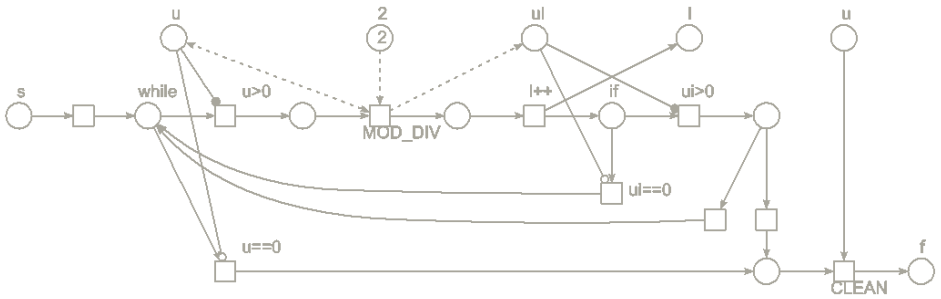
International Humanitarian University, Odessa, Ukraine

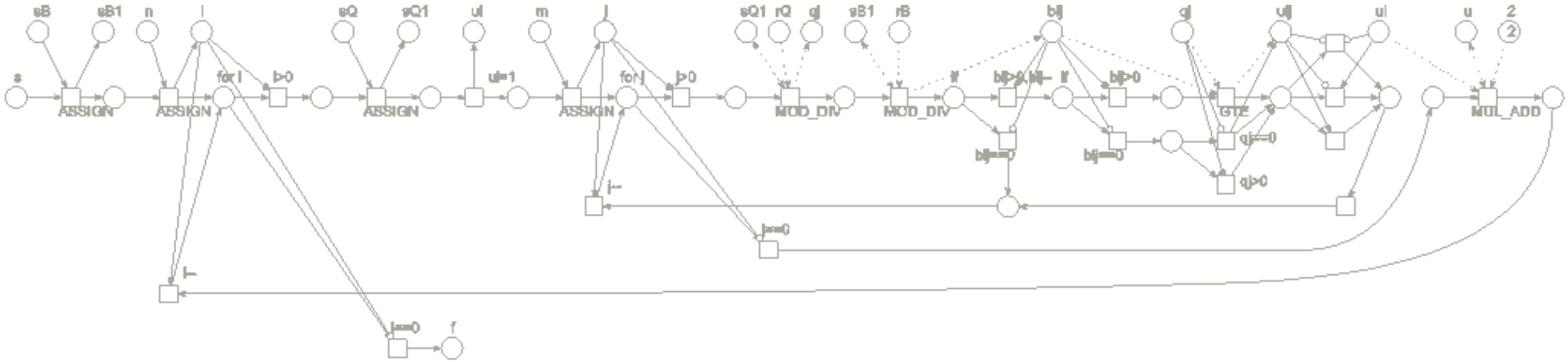
www.mgu.com.ua





Zaitsev D.A.

UNIVERSAL INHIBITOR PETRI NET

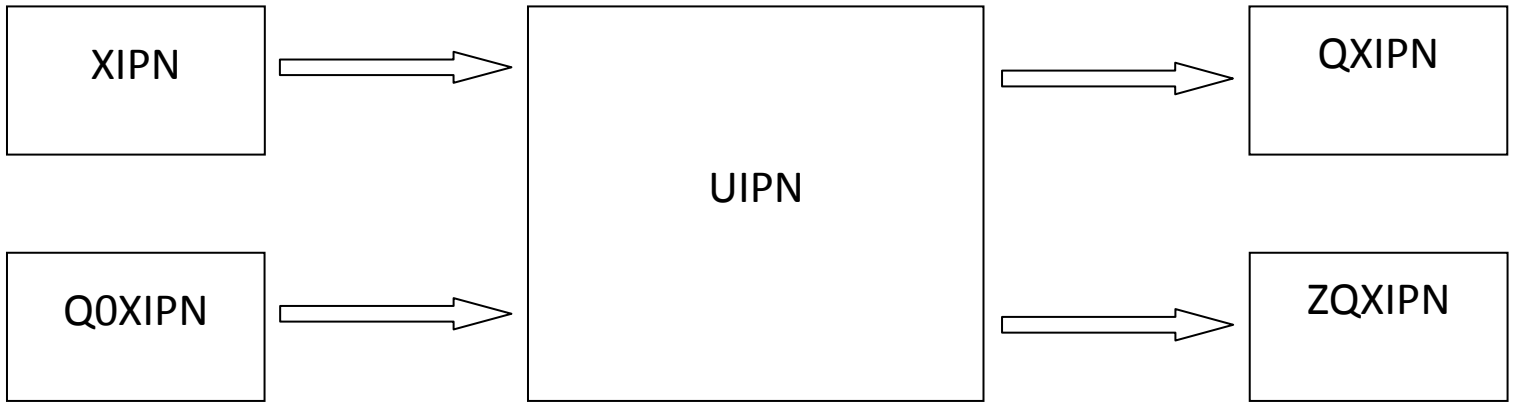


member.acm.org/~daze, daze.ho.ua

SOME BACKGROUND AND AFTERTHOGHTS

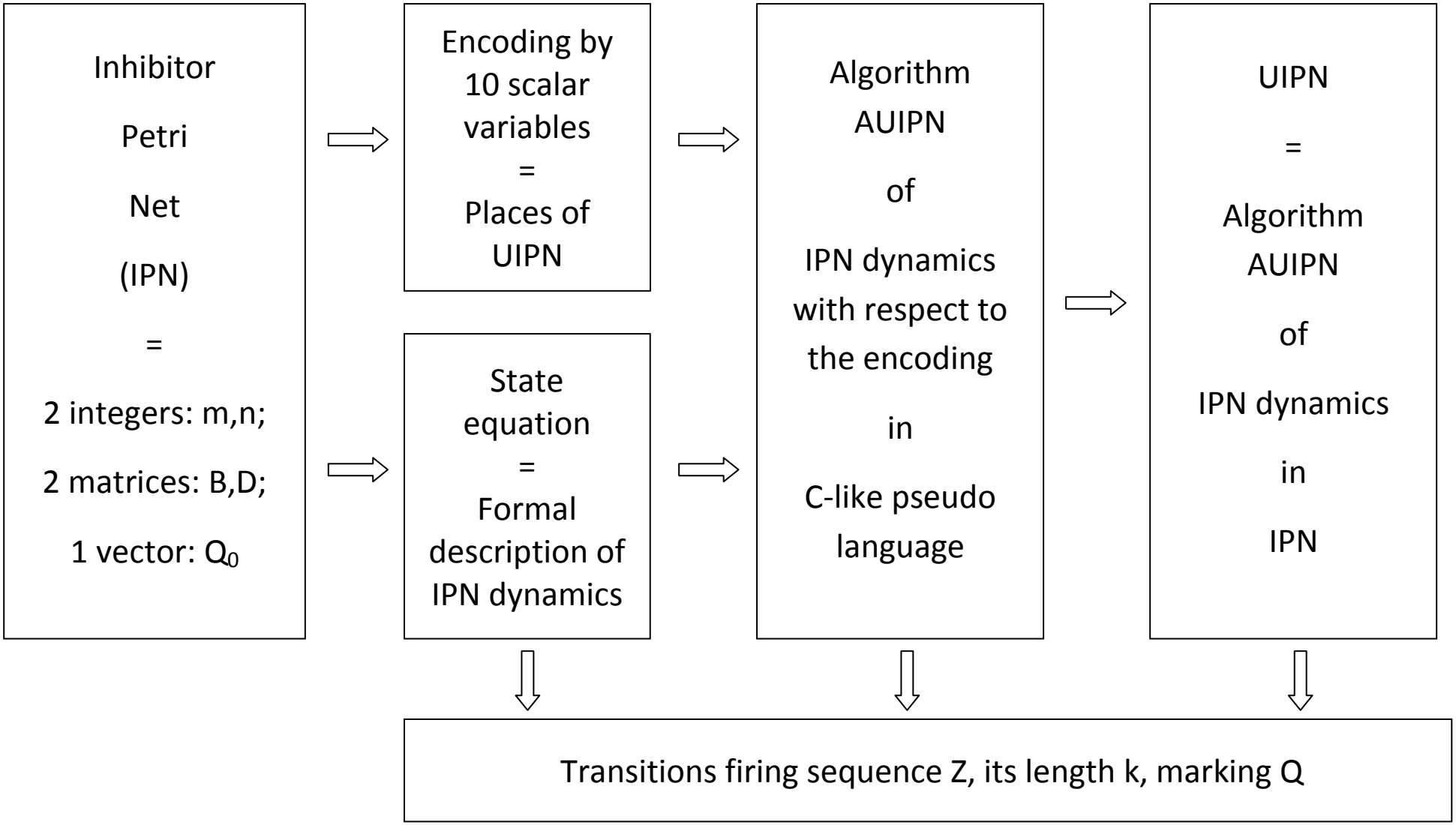
- Gottfried Wilhelm Leibniz (1646–1716) was musing on a Universal Solver of Tasks – Rechenmaschine
- David Hilbert presented his list of Problems (1900) and das Entscheidungsproblem (1928)
- Alan Turing formalized the intuitive concept of Algorithm in Turing Machine (1936) and constructed ***Universal Turing Machine that executes any given Turing Machine***
- Universal Turing Machine is a prototype of computers (since 1949)
- Karl Petri invented Petri nets (1962)
- ***Tilak Agerwala proved that Inhibitor Petri Net is a Universal Algorithmic System (1974)***
- Vadim Kotov – the representation of algorithms by Petri nets (1984)
- Anatoly Sleptsov – the representation of arithmetic and logic operations by Petri nets (1988)
- ***Why not construct Universal Inhibitor Petri Net?***
- It could be a new paradigm of computations.

THE CONCEPT OF A UNIVERSAL PETRI NET



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 PLOT


Inhibitor Petri Net (IPN) = 2 integers: m,n; 2 matrices: B,D; 1 vector: Q_0
Encoding by 10 scalar variables = Places of UIPN
State equation = Formal description of IPN dynamics
Algorithm AUIPN of IPN dynamics with respect to the encoding in C-like pseudo language
UIPN = Algorithm AUIPN of IPN dynamics in IPN
Transitions firing sequence Z, its length k, marking Q

FORMAL REPRESENTATION OF INHIBITOR PETRI NET

$N = (G, Q_0)$, where G – graph of net, and Q_0 – its initial marking

$G = (P, T, B, D)$, $P = \{p_1, .., p_m\}$ – places, $T = \{t_1, .., t_n\}$ – transitions;

$B: P \times T \to \mathbb{N} \cup \{-1\}$, $D: T \times P \to \mathbb{N}$ – arcs, $Q: P \to \mathbb{N}$ – marking.

Vector (matrix) representation: N=(m, n, B, D, Q_0),

$B = \|b_{i,j}\|, b_{i,j} = B(p_j, t_i)$, $D = \|d_{i,j}\|, d_{i,j} = D(t_i, p_j)$, $Q = \|q_j\|, q_j = Q(p_j)$.

Dynamics of net – state equation

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

ENCODING OF A VECTOR

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

Statement 1. The vector encoding function is injective.

Decoding:

$$q_j = \psi(s, j) = \left(s \; div \; r^j\right) mod \; r.$$

Recurrent encoding ($s = s \cdot r + q$):

$$s_j = s_{j-1} \cdot r + q_{m-1-j}, s_0 = q_{m-1}, j = \overline{1, m-1}, s = s_{m-1}. \quad (2)$$

Recurrent decoding ($q = s \; mod \; r, \; s = s \; div \; r$):

$$q_j = s_{m-1-j} \; mod \; r, s_{m-1-(j+1)} = s_{m-1-j} \; div \; r, s_{m-1} = s, \quad (3)$$

$$j = \overline{0, m-1} \,.$$

ENCODING OF A MATRIX

Matrix is represented as a vector with the expansion on lines

$$s = \varphi(A) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} r^{(m \cdot i + j)} \cdot a_{i,j}, r = \max_{i,j} a_{i,j} + 1.$$

Statement 2. The matrix encoding function is injective.

Decoding:

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

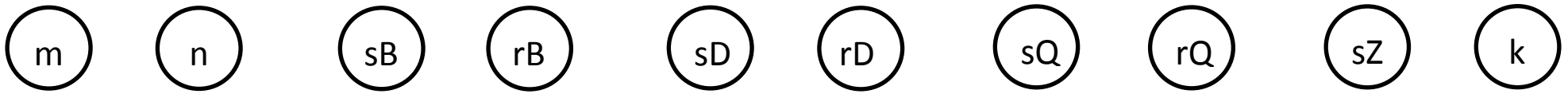
Recurrent encoding:

$$\begin{gathered} s_v = s_{v-1} \cdot r + a_{i,j}, s_0 = a_{n-1,m-1}, \\ v = \overline{1, n \cdot m - 1}, i = v \; div \; n, j = v \; mod \; n, s = s_{n \cdot m - 1}. \end{gathered} \quad (4)$$

Recurrent decoding:

$$\begin{gathered} a_{i,j} = s_{n \cdot m - 1 - v} \; mod \; r, s_{n \cdot m - 1 - (v+1)} = s_{n \cdot m - 1 - v} \; div \; r, s_{n \cdot m - 1} = s, \\ v = \overline{0, n \cdot m - 1}, i = v \; div \; n, j = v \; mod \; n. \end{gathered} \quad (5)$$

ENCODING OF AN INHIBITOR PETRI NET



m n sB rB sD rD sQ rQ sZ k

Petri net graph:

sB **– code of matrix** ***B*****,** ***rB*** **– radix for matrix** ***B*** **encoding**

sD **– code of matrix** ***D*****,** ***rD*** **– radix for matrix** ***D*** **encoding**

Matrix ***B*****: inhibitor arc – value 1, usual arc – value** $k + 1$**,** ***k*** **– multiplicity of arc.**

Marking:

sQ **– code of vector** ***Q*****,** ***rQ*** **– radix for vector** ***Q*** **encoding**

Transitions Firing Sequence:

sZ **– code of vector** ***Z*****,** ***n*** **– radix for vector** ***Z*** **encoding,** ***k*** **– its length**

Set of Enabled Transitions (binary indicators):

u **– code of vector** $\overline{u}$**, 2 – radix for vector** $\overline{u}$ **encoding**

ALGORITHM OF INHIBITOR PETRI NET EXECUTING

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

   inputXIPN();
   k=0; 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();
}
```

Notations:

- u – the code of enabled transitions indicator;
- l – the number of the firing transition;
- k – the number of the current step;
- CheckFire – checking the transitions enabling conditions;
- PickFire – the firing transition choice;
- Fire – the firing of the transition;
- NonDeterministic – nondeterministic choice of a number from the set {0,1};
- MUL_ADD – recurrent encoding (2);
- MOD_DIV – recurrent decoding (3).

PROCEDURES

```
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);
     }
}
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;
}
```

```
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,rD);
     }
     for(j=m; j>0; j--)
     {
          MOD_DIV(&qj,&sQ,rQ);
          MOD_DIV(&bij,&sB1,rB);
          if(bij>0) bij--;
          MOD_DIV(&dij,&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);
     }
}
void MUL_ADD(&x,y,z)
{
     (*x) = (*x) * y + z;
}
void MOD_DIV(&m,&x,y)
{
     (*m) = (*x) mod y;
     (*x) = (*x) div y;
}
```

JUSTIFICATION OF ALGORITHM AUIPN

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

– line 2, system (1), encoding (2) of vector $\overline{u}$

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

– line 3, system (1), decoding (3) of vector $\overline{u}$

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

– line 1, system (1), decoding/encoding (2,3), matrices B, D, vector Q

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

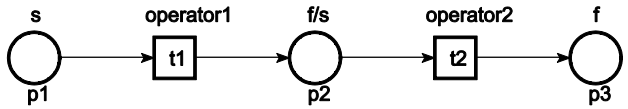
– it recalculates marking Q according to system (1),

– it stores the actual transitions firing sequence Z.

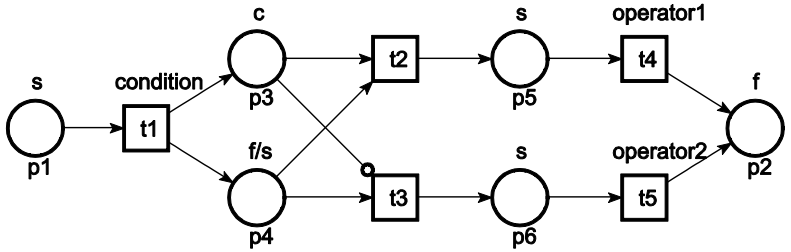
Theorem 2. **Algorithm AUIPN can be represented by an inhibitor Petri net.**

ENCODING OF ALGORITHMS BY INHIBITOR PETRI NET

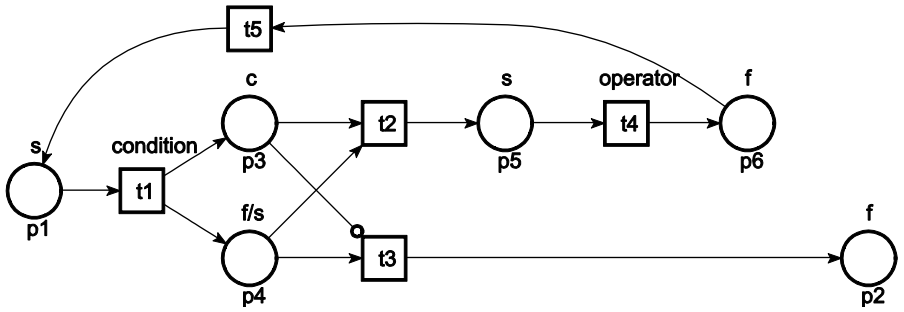
Lemma 4. **Algorithmic control constructions of the programming language can be encoded by inhibitor Petri net in the following way:**



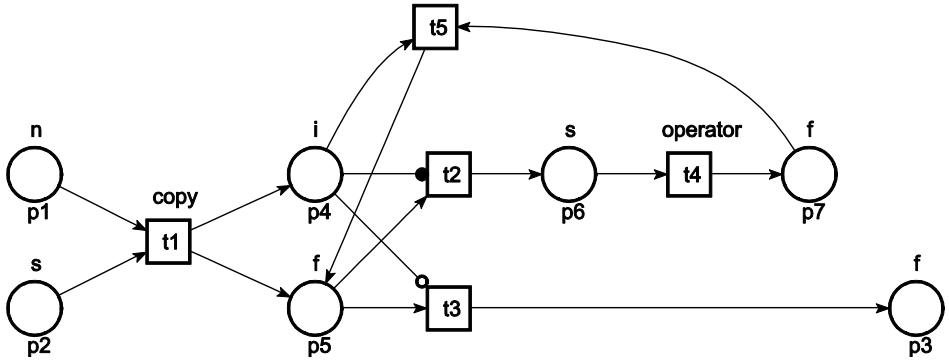
a) sequence –
operator1;operator2;



b) branch –
if(condition()) then operator1; else operator2;



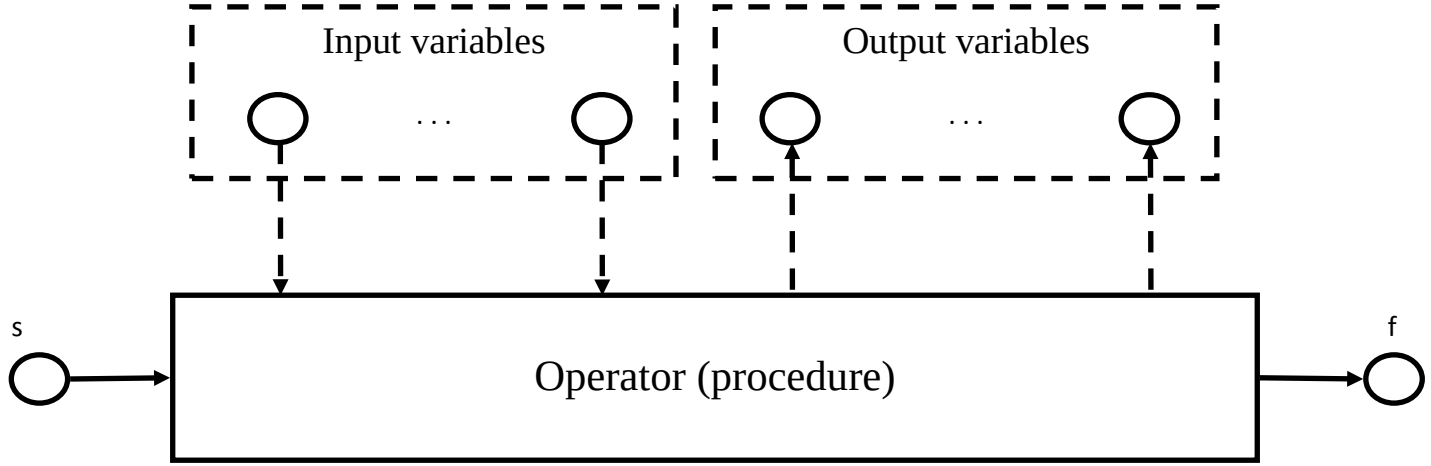
c) loop “while” – while(condition()) operator;



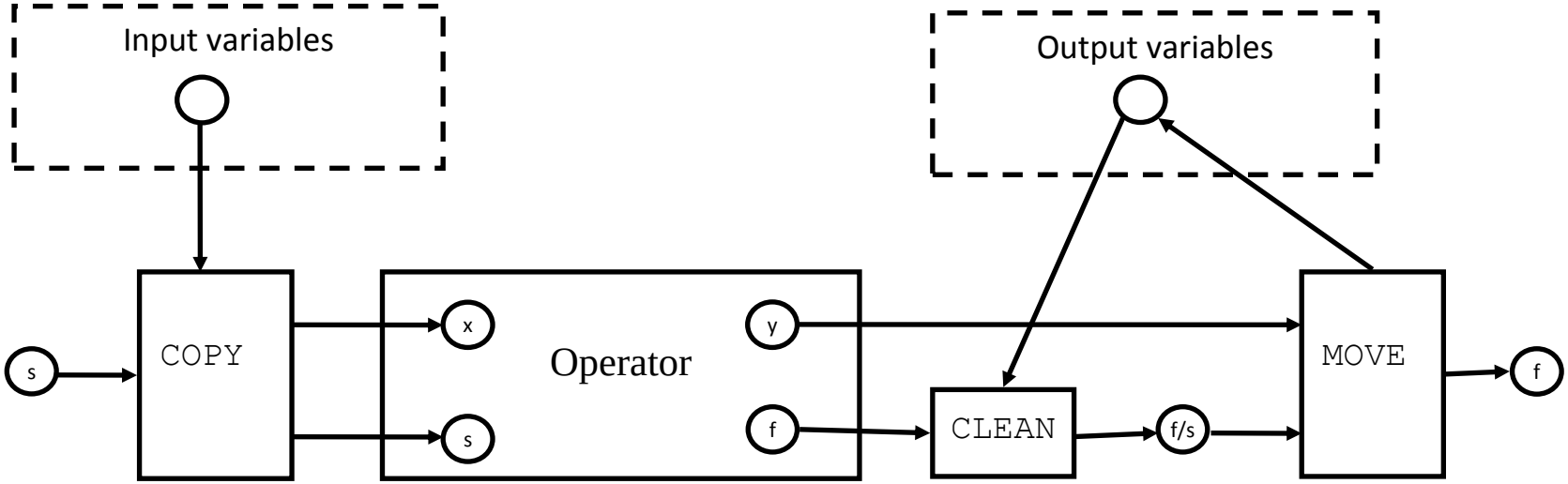
d) loop “for” – for(i=n;i>0;i--) operator;

WORK WITH VARIABLES

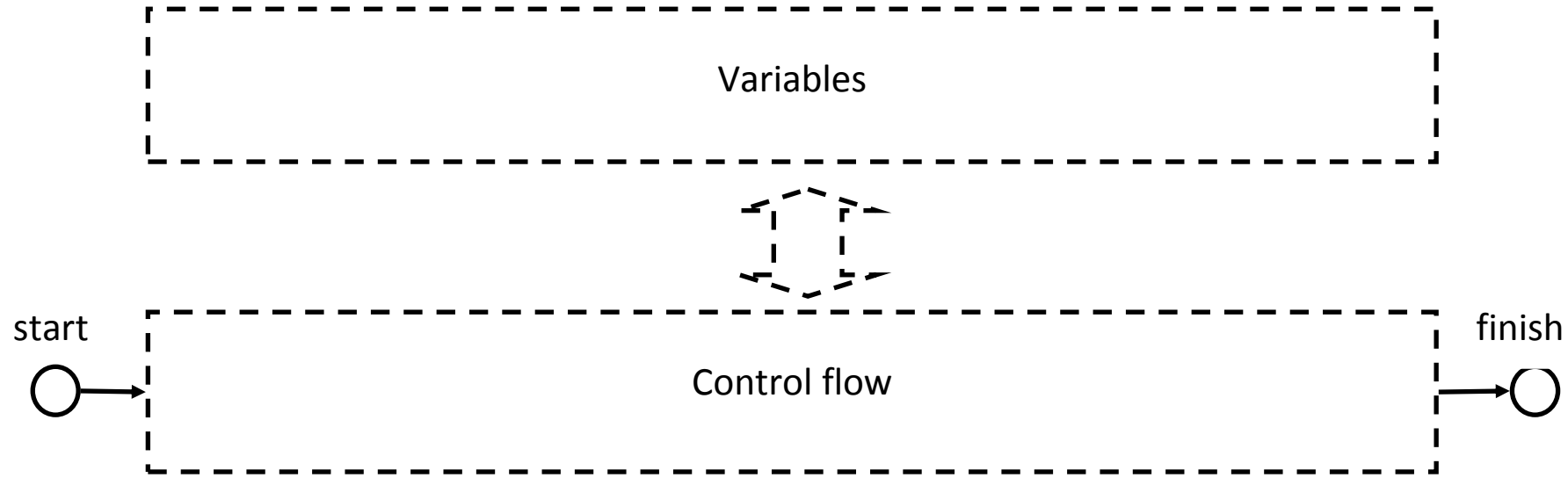
Representation of a programming language operator:



The forming of input and output variables:



OVERALL ORGANIZATION OF THE NET UIPN



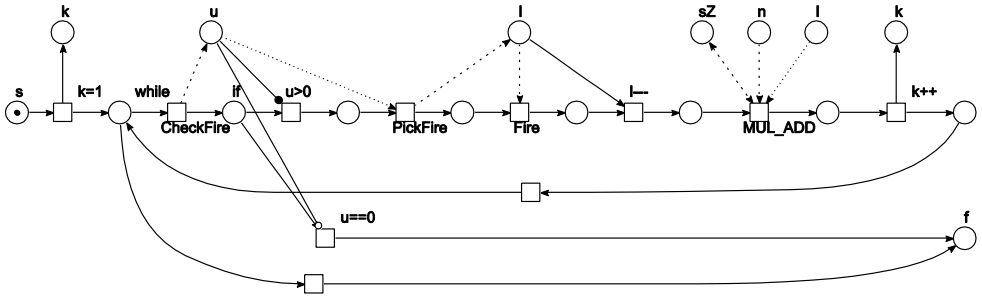
Auxiliary nets:

Work with variables: CLEAN – cleaning, COPY – copying, MOVE – moving, ASSIGN – assigning.

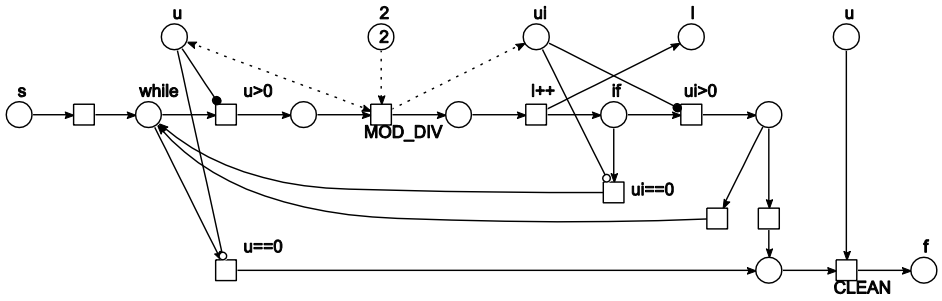
Arithmetic (logic) operations: ADD – addition, SUB – subtraction, MUL – multiplication, GTE – comparison, MAX – maximum.

Encoding of net: MUL_ADD – adding to the code, MOD_DIV – extracting from the code.

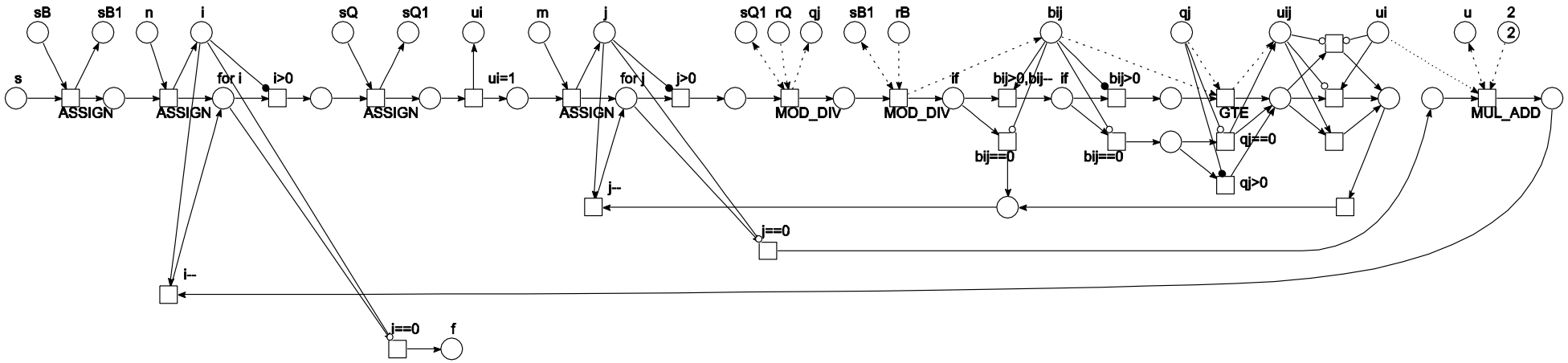
UNIVERSAL INHIBITOR PETRI NET UIPN



a) UIPN

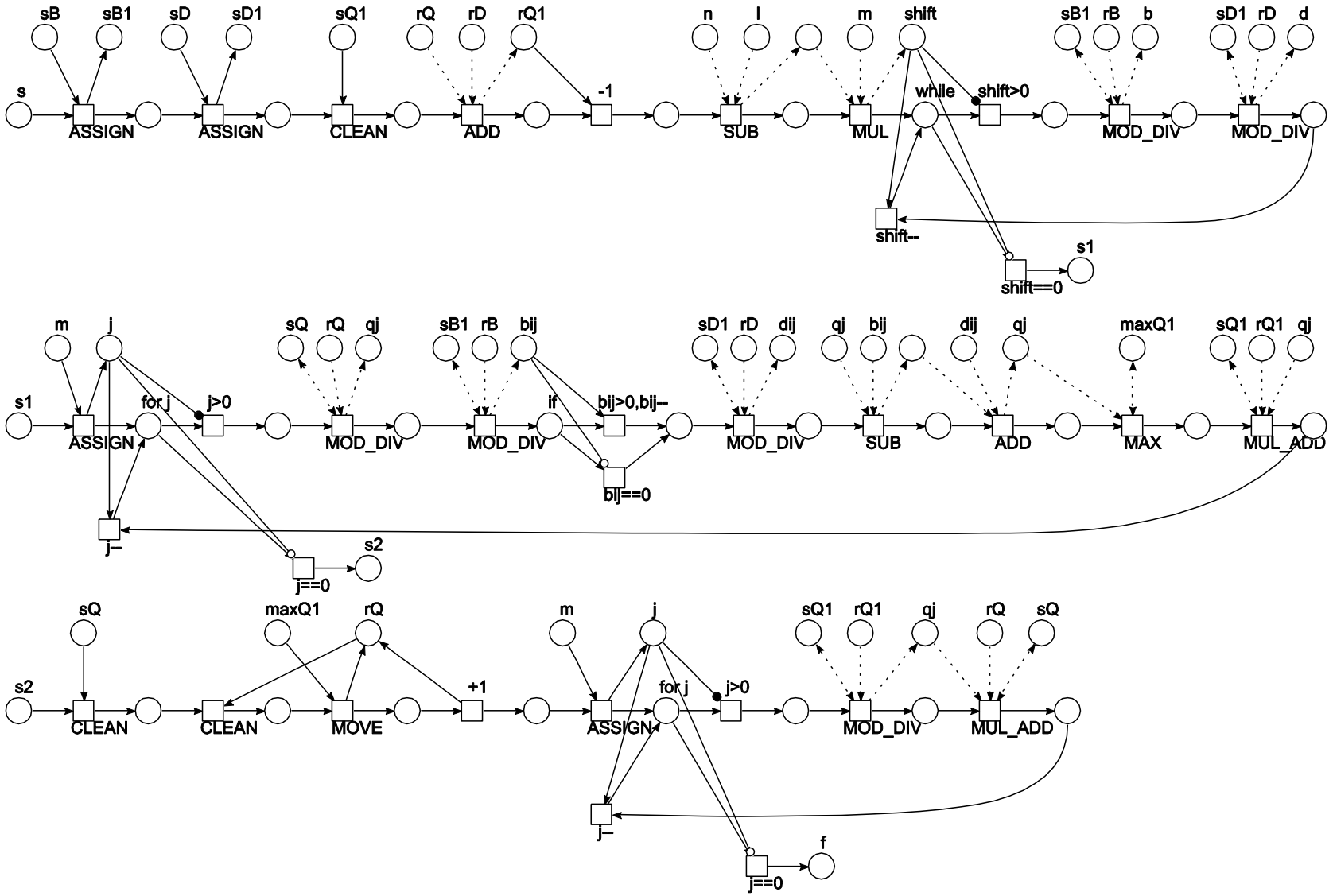


b) PickFire



c) CheckFire

UNIVERSAL INHIBITOR PETRI NET UIPN (COMPLETION)



d) Fire

LAYOUT OF THE PROOF

Lemma 5. **Auxiliary nets CLEAN, COPY, MOVE, ASSIGN, ADD, SUB, MUL, GTE, MAX, MUL_ADD, MOD_DIV implement the specified operations.**

– description and classification of all the enabled transitions firing sequences, composition of constructed nets by merging of contact places

Example ADD:

at $s = 0$, enabled transitions are absent

at $s = 1$, t_1 fires: $\sigma = t_1$, $p_6 = 1$

at $p_1 = x > 0$, t_2 is enabled, at $p_2 = y > 0$, t_3 is enabled

t_4 is enabled at $(p_1 = 0) \& (p_2 = 0)$ only

in an arbitrary order: t_2 fires x times, t_3 fires y times; then t_4 fires

firing sequence $\sigma = t_1 (t_2)^x (t_3)^y t_4$, the result: $p_4 = z = x + y$, $p_5 = f = 1$

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

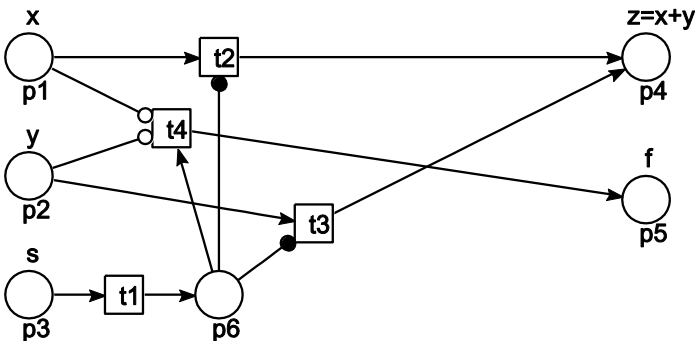
– corollary of theorem 1, lemmas 4,5.

CONCLUSIONS

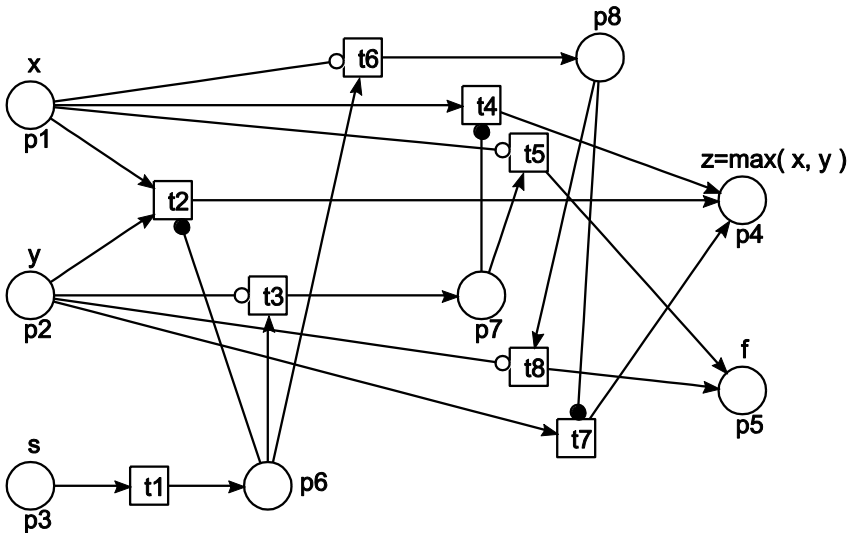
- **the Universal Inhibitor Petri Net was constructed that executes an arbitrary given inhibitor Petri net**
- **it is possible the constructing universal nets in other classes of Petri nets which are the universal algorithmic system: priority, synchronous, timed**
- **it is possible the combined constructing, for example: inhibitor net that executes an arbitrary synchronous net**
- **there is of a definite interest the constructing universal Petri net with the minimal number of places (transitions), the minimal value of the marking**
- **the Universal Inhibitor Petri Net could be considered as a new paradigm of computations**
- **Petri net processors (either software or hardware) will accomplish this paradigm**

EXAMPLES OF AUXILIARY NETS

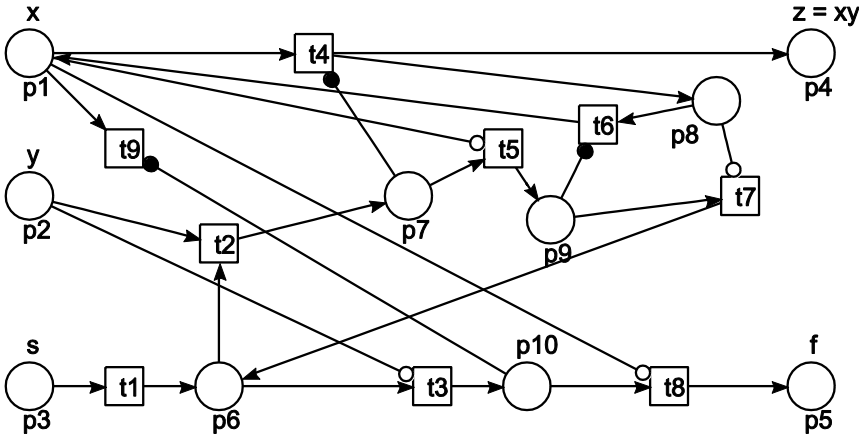
1) addition ($z = x + y$)
ADD::



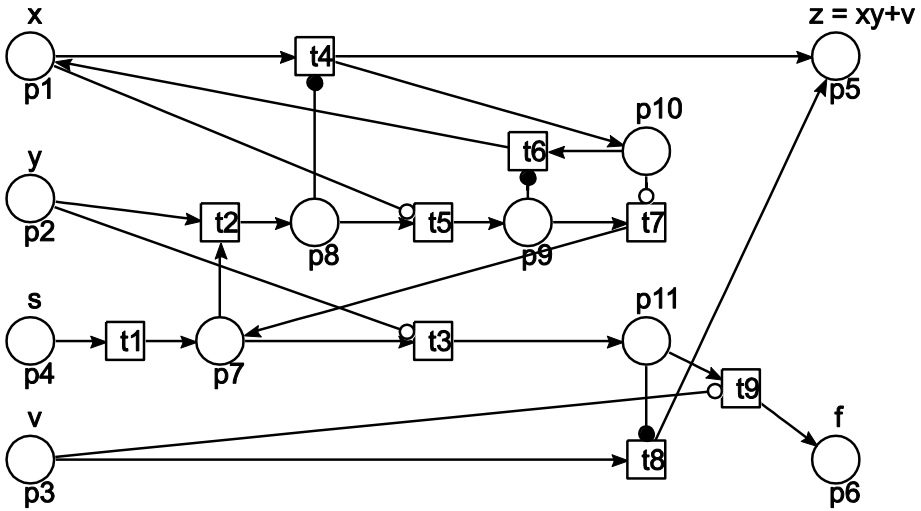
2) Maximum ($z = \max(x, y)$)
MAX::



3) Multiplication ($z = x \cdot y$)
MUL::



4) Addition to the code
MUL_ADD::



EXAMPLES OF NETS ENCODING

1) Petri net graph

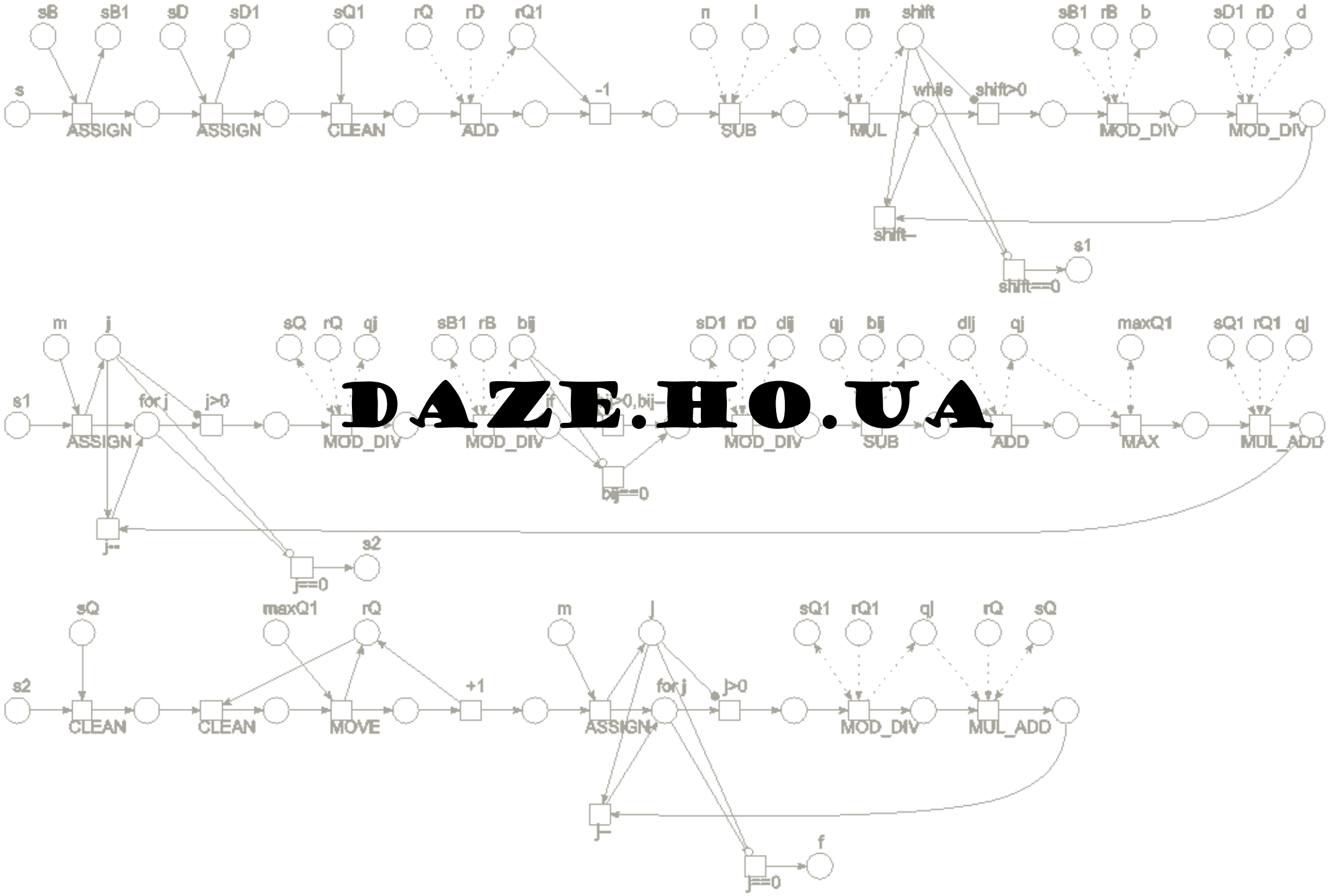
Net	m	n	sB	rB	sD	rD
ADD	6	4	21180169496	3	282946	2
MAX	8	8	254813592433189871074065241412	3	293862152152879368	2
MUL	10	9	64654907206110145566888903466348174395265 4	3	19352259085292454555975681	2

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


sB
sB1
sD
sD1
sQ1
rQ
rD
rQ1
n
l
m
shift
sB1
rB
b
sD1
rD
d
s
ASSIGN
ASSIGN
CLEAN
ADD
-1
SUB
MUL
while
shift>0
MOD_DIV
MOD_DIV
shift--
s1
shift==0
m
j
sQ
rQ
qj
sB1
rB
bij
sD1
rD
dlj
qj
bij
dlj
qj
maxQ1
sQ1
rQ1
qj
s1
ASSIGN
for j
j>0
MOD_DIV
MOD_DIV
if
bij>0,bij--
MOD_DIV
SUB
ADD
MAX
MUL_ADD
bij==0
j--
s2
j==0
sQ
maxQ1
rQ
m
j
sQ1
rQ1
qj
rQ
sQ
s2
CLEAN
CLEAN
MOVE
+1
ASSIGN
for j
j>0
MOD_DIV
MUL_ADD
j--
f
j==0