Journal of Automata, Languages and Combinatorics **u** (v) w, x–y
© Otto-von-Guericke-Universität Magdeburg

Small Universal Deterministic Petri Nets with Inhibitor Arcs

Artiom Alhazov
Institute of Mathematics and Computer Science, Academy of Sciences of Moldova
Academiei 5, MD-2028, Chişinău, Moldova
e-mail: `artiom@math.md`

Sergiu Ivanov
Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est
61, av. du gén. de Gaulle, 94010 Créteil, France
e-mail: `sergiu.ivanov@u-pec.fr`

Elisabeth Pelz
Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est
61, av. du gén. de Gaulle, 94010 Créteil, France
e-mail: `pelz@u-pec.fr`

and

Sergey Verlan
Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est
61, av. du gén. de Gaulle, 94010 Créteil, France
e-mail: `verlan@u-pec.fr`

ABSTRACT

This paper describes a series of small universal Petri nets with inhibitor arcs. Four parameters of descriptional complexity are considered: the number of places, transitions, inhibitor arcs, and the maximal degree of a transition. A number of techniques for reducing the values of these parameters, with special attention on places, are presented: we describe strongly universal Petri nets with 30, 21, 14, 11, and 5 places, and weakly universal Petri nets with similar parameters. We also show a universal Petri net with 2 inhibitor arcs only. Our investigation highlights several trade-offs. Due to equivalence of the corresponding models, our results can be immediately translated to multiset rewriting with forbidding conditions, to P systems with cooperative rules and inhibitors, or to vector addition systems.

Keywords: Petri nets; universality; descriptional complexity.

1. Introduction

Universality is an important concept in the theory of computation. The question of finding a universal computing device in the class of Turing machines was originally proposed by A. Turing himself in [22]. A universal Turing machine would be capable of simulating any other Turing machine $\mathcal{T}$: given a description of $\mathcal{T}$ and the encoding of the input tape contents, the universal machine would halt with tape contents which would correspond to the encoding of the output of $\mathcal{T}$ for the supplied input.

In a more general setting of an class arbitrary class $\mathfrak{C}$ of computing devices, the universality problem consists in finding such a *fixed* element $\mathcal{M}_0 \in \mathfrak{C}$ which would be able to simulate any other element $\mathcal{M} \in \mathfrak{C}$. More formally, if the result of running $\mathcal{M}$ with the input x is y (usually written as $M(x) = y$), then $y = f(\mathcal{M}_0(\langle g(\mathcal{M}'), h(x)\rangle))$, where g is the function enumerating $\mathfrak{C}$, $\langle\rangle$ is some couple encoding (e.g. the Cantor pairing function), while f and h are the decoding and encoding functions respectively. Although technically this definition does not impose any restriction on the encoding and the decoding functions, it is generally agreed that f and h should not be "too" sophisticated, so that it is mainly $\mathcal{M}_0$ who does the heavy lifting and not f and h. Since it is relatively common to rely on exponential coding when working with devices computing numbers, the functions $f(x) = \log_a(x)$ and $h(x) = b^x$, for some $a, b \in \mathbb{N}$ are often used (cf. [15, 24]).

If the considered class $\mathfrak{C}$ is a set of unary non-negative functions, then the encoding and decoding functions are not mandatory. In this paper, we will adhere to the terminology established by I. Korec in [13] and call the element $\mathcal{M}_0$ defined as above *weakly* universal (or just universal). In case the functions f and h are additionally required to be identities, $\mathcal{M}_0$ will be referred to as *strongly* universal. Hence, the strong universality permits to capture the situations when the encoding does not alter the power of the device. For example, 2-register machines are weakly universal [15], but they cannot be strongly universal as they cannot compute even the square function [3, 20].

As a further development on the question of universality, C. Shannon [21] considered finding the *smallest* possible universal Turing machine, where the size is essentially given by the sizes of the alphabets of symbols and states. A series of important results concerning this direction were obtained [14, 19, 23]. For an overview of the recent results the reader is referred to [16]. Small universal devices are of considerable theoretical importance since they indicate the minimal choice ingredients sufficient for achieving computational completeness.

Register machines were shown to be universal and it is known that already three registers suffice for strong universality [11, 15]. In 1996, I. Korec described a number of universal register machines with considerably fewer instructions than were known to be needed for universality before [13]. Based on these results, several universal constructions for multiset rewriting models were proposed, e.g. with maximally parallel rule execution [2] or with weighted inhibitors [6]. Since such models are computationally equivalent to vector additions systems [5] and Petri nets, corresponding constructions can be immediately translated between all three classes. Yet, no explicit descriptions of universal Petri nets have been proposed until the recent work [26] by D. Zaitsev in which a universal Petri net with 14 places and 29 transitions is described. This work relies on both inhibitor arcs and priorities, while in [25] a universal Petri net relying on inhibitor arcs only with 500 places and as many transitions it is shown.

In this article we focus on Petri nets with (non-weighted) inhibitor arcs and without priorities, since the two extensions are equivalent as far as computational power is concerned. We systematically investigate such Petri nets working in the classical sequential semantics and measure the following four fundamental parameters of the net: the number of places p, of transitions t, of inhibitor arcs h, and the maximal degree of a transition d. The tuple (p, t, h, d) will be referred to as the *size* of a Petri net. Remark that, from the point of view of multiset rewriting, these parameters correspond to the size of the alphabet, the number of rules, the number of forbidding conditions, and the maximal size of a rule, respectively. We therefore consider that these properties are essential to the characterization of a Petri net as a computational device. We construct several universal nets whose parameters are summarized in Tables 2 and 3. The appendix of [10] contains incidence tables of some of the nets.

The paper is organized as follows. Section 2 recalls some preliminaries on the models used in this paper. Section 3 introduces the notion of *generalized register machine* that is used as an intermediary step for many further universal Petri net constructions. Section 4 gives constructions for several universal Petri nets following the goal of minimization of a single descriptional complexity parameter. Finally, Section 5 focusses on the design of reusable components that can be combined to achieve a register machine simulation by Petri nets. Such a design permits to keep relatively low values for all 4 parameters. We would also like to remark that this paper is an extended version of [8].

2. Preliminaries

In this section we will briefly define the concepts we use in the rest of the paper. We will first recall some basic notions from the formal language theory.

Any non-empty finite set of symbols V is called an *alphabet*. A finite word over V is any finite sequence of symbols from V. The length of a word is written as $|w|$; the notation $|w|_a$ refers to the number of occurrences of the symbol a in w. The set of all words over V of length n is denoted by V^n. By definition, the only word of zero length is the empty word λ: $V^0 = \{\lambda\}$. The set of all finite words over V is denoted by $V^* = \cup_{n \in \mathbb{N}} V^n$, and the set of non-empty finite words by $V^+ = \cup_{n \in \mathbb{N}^+} V^n = V \setminus \{\lambda\}$. Any subset of V^* is called a *language* over V.

2.1. Register Machines

A (deterministic) *register machine* is defined as a 5-tuple $M = (Q, R, q_0, q_f, P)$, where Q is a set of states, $R = \{R_1, \ldots, R_k\}$ is the set of registers, $q_0 \in Q$ is the initial state, $q_f \in Q$ is the final state, and P is a set of instructions of the following three forms:

- *(increment)* $(p, A(i), q)$, where $p, q \in Q$, $R_i \in R$: being in state p, increment register R_i and go to state q;
- *(zero-check-and-decrement)* $(p, S(i), q, s)$, where $p, q, s \in Q, R_i \in R$: being in state p, try decrementing register R_i and go to q if successful or to s otherwise;
- *(stop)* $(q_f, Stop)$: halt the execution; associated only to final state q_f.

In some sources (e.g., [9, 10, 13]), the increment is written as RiP, and the zero-check-and-decrement instruction as $RiZM$. The work [13] also uses other instruction types, like a separate decrement (p, RiM, q), which corresponds to $(p, S(i), q, q)$, or a separate zero-check (p, Ri, q, s), which works like zero-check-and-decrement, but does not modify the register: it corresponds to the pair of instructions $(p, S(i), q', s)$ and $(q', A(i), q)$.

Sometimes *non-deterministic* register machines are considered; in this case, the increment instruction is defined to have the form $(p, A(i), q, q')$. Thus, after carrying out the increment in state p, the machine can non-deterministically choose between states q and q'.

Register machines are often represented graphically in flowchart notation; in this case the instructions checking a condition on a register are drawn as diamonds, while non-conditional instructions as rectangles.

A configuration of a register machine is given by $(q, n_1, \ldots, n_k)$, where $q \in Q$ and $n_i \in \mathbb{N}, 1 \leq i \leq k$, describe the current state of the machine as well as the contents of all of its registers. A transition of the register machine consists in updating or checking the value of a register according to an instruction of one of the types above and in changing the current state to another one. We say that the machine stops if it reaches the state q_f. We say that M computes a value $y \in \mathbb{N}$ on the input $x_1, \ldots, x_n$, $x_i \in \mathbb{N}$, $1 \leq i \leq n \leq k$, if, starting from the initial configuration $(q_0, x_1, \ldots, x_n, 0, \ldots, 0)$, it reaches the final configuration $(q_f, y, 0, \ldots, 0)$.

In this work we will rely heavily on universal register machines constructed by Ivan Korec in [13]. We give below the program of U_{22} – the strongly universal register machine with 22 commands.

$(q_1, R1ZM, q_3, q_6)$	$(q_3, R7P, q_1)$	$(q_4, R5ZM, q_6, q_7)$
$(q_6, R6P, q_4)$	$(q_7, R6ZM, q_9, q_4)$	$(q_9, R5P, q_{10})$
$(q_{10}, R7ZM, q_{12}, q_{13})$	$(q_{12}, R1P, q_7)$	$(q_{13}, R6ZM, q_{33}, q_1)$
$(q_{33}, R6P, q_{14})$	$(q_{14}, R4ZM, q_1, q_{16})$	$(q_{16}, R5ZM, q_{18}, q_{23})$
$(q_{18}, R5ZM, q_{20}, q_{27})$	$(q_{20}, R5ZM, q_{22}, q_{30})$	$(q_{22}, R4P, q_{16})$
$(q_{23}, R2ZM, q_{32}, q_{25})$	$(q_{25}, R0ZM, q_1, q_{32})$	$(q_{27}, R3ZM, q_{32}, q_1)$
$(q_{29}, R0P, q_1)$	$(q_{30}, R2P, q_{31})$	$(q_{31}, R3P, q_{32})$
$(q_{32}, R4ZM, q_1, q_f)$	$(q_f, STOP)$	

2.2. Universal 2- and 3-Register Machines

We recall that there exist weakly universal register machines with two registers only, and that strong universality for unary functions can be achieved by adding a third input/output register. This result was proved by Marvin Minsky in [15], a work which also gives the actual algorithm for simulating a register machine with any number of registers using only two (respectively, three) registers. In this section we will discuss the application of this algorithm to universal register machines constructed by Ivan Korec [13] in order to give a concrete description of 2- and 3-register machines. We remark that, even though Minsky's approach has been known for a long time, we could not find concrete constructions presented anywhere.

Consider a register machine $M = (Q, R, q_0, q_f, P)$. Minsky's construction is based on exponential coding of the values of all registers in R as one number. The values of the k registers in a configuration of M given by the vector $(n_1, \ldots, n_k)$ can be represented as the number $\rho = p_1^{n_1} \times \ldots \times p_k^{n_k}$, where $p_1, \ldots, p_k$ are different prime numbers. Then, incrementing the register R_i in this coding corresponds to multiplying ρ by p_i, while decrementing the same register corresponds to dividing ρ by p_i. For more details on this simulation approach we refer the reader to Minsky's work [15]. To achieve strong universality for unary functions, a third input/output register is required to carry out exponentiation and logarithm, as it is known that without encoding only simple functions can be computed using only 2 registers [3].

The strategy for achieving weak universality with two registers can be applied directly to constructing the weakly universal 2-register machine U_2 which simulates Korec's weakly universal U_{20}. This 2-register machine has 112 decrement and 165 increment instructions: 278 states all in all (including a final state). Simulating Korec's strongly universal U_{22} using two registers, and further adding the coding of input and the decoding of output, allows building the strongly universal 3-register machine U_3 with 146 decrement and 221 increment instructions: 368 states in total. Both register machines use register R_0 to store the exponentially-coded values of the simulated machine, and register R_1 to keep the intermediate results. The input of U_2 should thus be provided in coded form in R_0. The register machine U_3, on the other hand, reads its input from and writes its output to the third register, R_2. Full programs of U_2 and U_3 are provided in the appendix of [7].

The work [1] takes this reasoning even further by pointing out that both the cases of an increment and a successful decrement finalize by copying the result from R_1 to R_0. In many situations this "recopying" of the result can be avoided by designing the following operation to work on R_1 instead of R_0. However, only some of the presented constructions can be directly used for our goal of the Petri net simulation.

An important remark with regard to the strong universality of 3-register machines is due here: since such machines use one register for input, they are only able to directly simulate unary partial recursive functions. Nevertheless, Section 9 of [13] describes a way to construct register machines simulating n-ary partial recursive functions; the machines use a coding to store the values of the n arguments in one of the working registers. This approach can be naturally adapted to 2-register machines to obtain strongly universal register machines with n input registers, read by successive decrements at the start of the computation, and which only have two working registers.

We remark that Section 2 of [12] describes the construction of a universal 3-register machine with 130 states. However in this paper compound instructions are assigned to single states (e.g. any increment of a register by m is treated as a single instruction). Writing out such instructions in terms of elementary register machine commands as we do in [7], however, would yield more than 450 instructions.

2.3. Petri Nets with Inhibitor Arcs

A *Petri net with inhibitor arcs* is a construct $N = (\mathcal{P}, \mathcal{T}, W, M_0)$ where $\mathcal{P}$ is a finite set of *places*, $\mathcal{T}$ is a finite set of *transitions*, with $\mathcal{P} \cap \mathcal{T} = \emptyset$, $W : (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P}) \to \mathbb{N} \cup \{-1\}$ is the weight function, and M_0 is a multiset over $\mathcal{P}$ called the initial marking.

Petri nets are usually represented by diagrams in which places are drawn as circles, transitions as squares annotated with their locations, and a directed arc (X, Y) is added between X and Y if $W(X, Y) \geq 1$. The weight of the arc will be explicitly written if it is 2 or more. Arcs of weight -1 are called *inhibitor* arcs and will be drawn with a small circle on the side of the transition.

The *degree* of a transition T is defined as the sum of the weights of the incoming and outgoing arcs involved with it plus the number of inhibitor arcs:

$$deg(T) = \sum_{P \in \mathcal{P}} \big|W(P,T)\big| + \big|W(T,P)\big|.$$

Note that the degree is not the number of weighted arcs adjacent to the transition, but rather the number of single arcs they represent.

Given a Petri net N, the pre- and post-multiset of a transition T are respectively the multiset $pre_{N(T)}$ and the multiset $post_{N(T)}$ such that, for all $P \in \mathcal{P}$, for which $W(P,T) \geq 0$, $pre_{N(T)}(P) = W(P,T)$ and $post_{N(T)}(P) = W(T,P)$. A configuration of N, which is called a *marking*, is a multiset M over $\mathcal{P}$; in particular, for every $P \in \mathcal{P}$, $M(P)$ represents the number of tokens inside place P. A transition T is enabled at a marking M if the multiset $pre_{N(T)}$ is contained in the multiset M and all inhibitor places P (such that $W(P,T) = -1$) are empty. A transition T enabled at marking M can fire and produce a new marking M' such that $M' = M - pre_{N(T)} + post_{N(T)}$ (i.e. for every place $P \in \mathcal{P}$, the firing transition T consumes $pre_{N(T)}(P)$ tokens and produces $post_{N(T)}(P)$ tokens). We denote this by $M \xrightarrow{T} M'$.

For the purposes of this work, we define a special subtype of Petri nets which can execute computations (e.g. compute partially recursive functions). In such a net some distinguished places $I_1, \ldots, I_k$ from $\mathcal{P}$ will be called input places and another one, $I_0 \in \mathcal{P}$, will be called the output place. The computation of the net N on the input vector $(n_1, \ldots, n_k)$ starts with the initial marking M'_0 such that $M'_0(I_j) = n_j$, and $M'_0(x) = M_0(x)$, for all $x \neq I_j$, $1 \leq j \leq k$. This net will evolve by firing transitions until deadlock occurs in some marking M_f, i.e. until no transition is enabled in M_f. Thus we have $M'_0 \to^* M_f$ and there are no M'_f and $T \in \mathcal{T}$ such that $M_f \xrightarrow{T} M'_f$. The result of the computation of N on the vector $(n_1, \ldots, n_k)$, denoted by $\Phi^k_N(n_1, \ldots, n_k)$, is defined as $M_f(I_0)$, i.e. the number of tokens in place I_0 in the final state. Since in the general case Petri nets are non-deterministic, the function Φ^k_N may be considered to compute sets of numbers.

If, for any reachable marking M of a Petri net N, there is at most one transition T and one marking M' such that $M \xrightarrow{T} M'$, the Petri net is called *deterministic*. This corresponds to labeled deterministic Petri nets in which all transitions are labeled with the same symbol [17]. Otherwise the Petri net is called *non-deterministic*.

We define the *size* of a Petri net to be the vector (p, t, h, d) where p is the number of places, t is the number of transitions, h is the number of inhibitor arcs, and d is the maximal degree of a transition. These parameters provide fundamental information about the structure of the net and can be further used to reason about its other features (e.g., the average number of inhibitor arcs per transition). Moreover, each parameter has a direct equivalent in the multiset rewriting interpretation of Petri nets as the cardinality of the alphabet, number of rules, inhibitors and maximal rule size.

3. Generalized Register Machines

In this section we discuss an extension of the construct of a register machine which originates in the fundamental similarities of this computing model with multiset rewriting systems and Petri nets.

3.1. Definition and Semantics

A *generalised register machine* is a construct $M = (R, G, q_0, F, z, nz, add, sub)$, where $R = \{1, \dots, k\}$ is a set of register numbers, $G = (Q, E, s, t)$ is a directed multigraph of states Q and arcs (transitions) E, $q_0 \in Q$ is the initial state, $F \subseteq Q$ comprises the final states, while $z, nz : E \to 2^R$, $add : E \to R^*$, and $sub : E \to 2^R$ are functions defining the registers which should be zero or non-zero for a transition to be activated, as well as the registers which should be decremented or incremented during the transition, respectively. We require that, for any $e \in E$, $z(e) \cap nz(e) = \emptyset$, and that $sub(e) \subseteq nz(e)$, i.e. only those registers which are required to be non-zero are decremented. Remark that $add(e)$ is a multiset, which means that multiple increments of the same register can be carried out in a single transition.

A configuration of a generalised register machine is the tuple $(q, n_1, \dots, n_k)$, where, just as in the case of conventional register machines, $q \in Q$ is the current state of M, and $n_i \in \mathbb{N}$, $1 \leq i \leq k$, are the values of the registers. We say that there is a transition from configuration $C = (q, n_1, \dots, n_k)$ to $C' = (q', n'_1, \dots, n'_k)$ if G contains an edge e such that $s(e) = q$, $t(e) = q'$, the conditions represented by $z(e)$ and $nz(e)$ hold in C:

$$\forall i \in z(e) \,.\, n_i = 0 \quad \text{and} \quad \forall i \in nz(e) \,.\, n_i \neq 0,$$

and the values of registers in C' can be obtained by applying the instructions prescribed by $s_e = sub(e)$ and $a_e = add(e)$:

$$\forall i \in R \,.\, n'_i = n_i - s_e(i) + a_e(i),$$

where $x(i)$ is the number of occurrences of i in the multiset x, and the set s_e is treated as a multiset. The arc e is said to be enabled in configuration C.

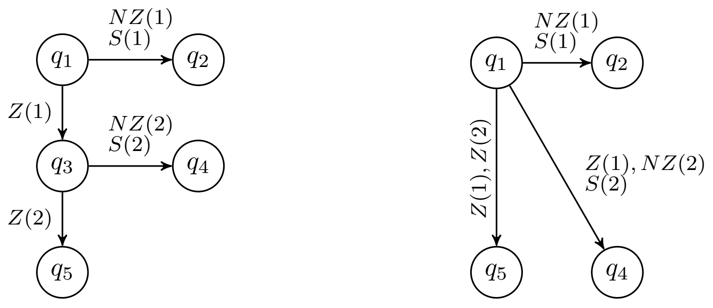
We will say that a generalised register machine is *deterministic* if, in every configuration $C = (q, n_1, \dots, n_k)$, $q \in Q \setminus F$, there is precisely one enabled arc, and in any configuration $C = (q_f, n_1, \dots, n_k)$, $q_f \in F$, no arcs are enabled.

In the graphical representation of the labels of an arc e of a generalised register machine, we use the symbol $Z(i)$ if $i \in z(e)$, $NZ(i)$ if $i \in nz(e)$, $S(i)$ if $i \in sub(e)$, and $A^j(i)$ if the multiplicity of the register number i in the multiset $add(e)$ is j. If $j = 1$, we will just write $A(i)$.

It follows from the definition that any register machine based on the increment and decrement-and-zero-check operations can be directly transformed into a generalised register machine; in particular, U_{22} and U_{20} from [13] can be directly seen as generalised register machines. Some more care should be taken in case the pure decrement instruction is used without a prior zero test, because in [13, Section 1] this instruction is defined to not modify the contents of the register if it is already empty. This corresponds to a decrement-and-zero-check instruction $(p, S(i), q, q)$, which moves the machine into state q no matter what the value of the register was, and should thus be simulated using two arcs of a generalised register machine having complementary sets of conditions on the register i.

3.2. State Compression and Universality

As we have seen in the previous subsection, generalised register machines are capable of performing several register tests and modifications at a single time. This property can be used to reduce the number of states at the expense of having more complex transitions. Consider for example the generalized register machine shown in Figure 1a; it could be described with the instructions $(q_1, S(1), q_2, q_3)$ and $(q_2, S(j), q_4, q_5)$. The generalized register machine in Figure 1b performs the same operations, but uses one state less. The idea is that, instead of checking that register 1 is zero on the transition from q_1 to q_3, and then checking that register 2 is non-zero and decrementing it while moving to state q_4, the register machine can do all of these checks and operations in a single direct transition from q_1 to q_4. We will refer to the procedure of removing states while conserving the same behaviour as *state compression*.



(a) Two decrements (b) Compressed version

Figure 1: State compression for successive $S(1)$ and $S(2)$.

While many states can be reduced away in this fashion, not all of them can. To formally define the circumstances under which a state may be removed, consider a state q, the set of edges which end in q: $pred(q) = \{e \in E \mid t(e) = q\}$, and another set of edges which start at q: $succ(q) = \{e \in E \mid s(e) = q\}$. Then q is compressible if all of the following conditions hold:

- no arc going into q modifies a register which is involved in the conditions of an arc leaving q, that is, for every $e_{pred} \in pred(q)$ and $e_{succ} \in succ(q)$,

$$\big(supp(add(e_{pred})) \cup sub(e_{pred})\big) \cap \big(z(e_{succ}) \cup nz(e_{succ})\big) = \emptyset;$$

- q has no loop arcs (i.e. arcs which do not move the machine away from state q):

$$\nexists e \in E \,.\, s(e) = t(e) = q.$$

Note that, since all registers which are decremented by an arc are required to be checked for being non-zero by the same arc, the former condition implies that a state, for which an incoming arc decrements a register and an outgoing arc decrements the same register, is not compressible.

To compress away a state q, we take all pairs of edges $e_{pred} \in pred(q)$ and $e_{succ} \in succ(q)$ and pick the pairs which do not check for conflicting conditions, that is:

$$z(e_{pred}) \cap nz(e_{succ}) = nz(e_{pred}) \cap z(e_{succ}) = \emptyset.$$

Then, for every such pair we add a new arc $e : q_{pred} \to q_{succ}$, putting together all the conditions and operations of the arcs e_{pred} and e_{succ}:

$$\begin{aligned} z(e) &= \quad z(e_{pred}) \cup z(e_{succ}), \\ add(e) &= add(e_{pred}) + add(e_{succ}), \\ nz(e) &= \; nz(e_{pred}) \cup nz(e_{succ}), \\ sub(e) &= sub(e_{pred}) \cup sub(e_{succ}). \end{aligned}$$

The state q and all arcs having it as source or target are then removed.

The *state compression* algorithm is defined as the iterative reduction of compressible states. It allows to compress the original register machine U_{22} to a generalized register machine with 7 states, including a *Stop* state. We will refer to this generalized machine as U_7; its program is shown in Table 1. Similarly, the register machine U_{20} can be compressed to a weakly universal generalised register machine with 7 states, which we will call U'_7; the program of this machine is given in the appendix of [7]. Both machines start in state q_1 with input in register 2 (corresponding to R_2 in U_{22} and U_{20}), and place the result into register 0 (corresponding to R_0 in U_{22} and U_{20}).

Table 1: The program of the universal generalised register machine U_7

q_i	q_j	Conditions	Operations
q_1	q_1	NZ(1)	S(1), A(7)
q_1	q_4	Z(1)	A(6)
q_4	q_4	Z(5), Z(6)	
q_4	q_4	NZ(5)	S(5), A(6)
q_4	q_{10}	Z(5), NZ(6)	A(5), S(6)
q_{10}	q_1	Z(6), Z(7)	
q_{10}	q_1	NZ(4), NZ(6), Z(7)	S(4)
q_{10}	q_4	Z(6), NZ(7)	A(1), S(7)
q_{10}	q_{10}	NZ(6), NZ(7)	A(1), A(5), S(6), S(7)
q_{10}	q_{16}	Z(4), NZ(6), Z(7)	
q_{16}	q_1	NZ(0), Z(2), Z(5)	S(0)
q_{16}	q_1	NZ(2), NZ(4), Z(5)	S(2), S(4)
q_{16}	q_1	Z(0), Z(2), NZ(4), Z(5)	S(4)
q_{16}	q_{18}	NZ(5)	S(5)
q_{16}	q_{34}	Z(0), Z(2), Z(4), Z(5)	
q_{16}	q_{34}	NZ(2), Z(4), Z(5)	S(2)
q_{18}	q_1	Z(3), Z(5)	A(0)
q_{18}	q_1	NZ(3), NZ(4), Z(5)	S(3), S(4)
q_{18}	q_{20}	NZ(5)	S(5)
q_{18}	q_{34}	NZ(3), Z(4), Z(5)	S(3)
q_{20}	q_1	NZ(4), Z(5)	A(2), A(3), S(4)
q_{20}	q_{16}	NZ(5)	A(4), S(5)
q_{20}	*Stop*	Z(4), Z(5)	A(2), A(3)

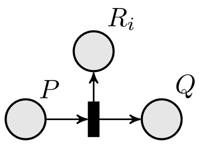
4. Small Universal Petri Nets

In this section, we will describe several small-size Petri nets with inhibitor arcs which achieve strong and weak universality. We will then evaluate the complexity parameters of the obtained nets: the number of places, the number of transitions, the number of inhibitor arcs and the maximal degree of transitions. In this section our main goal is to give constructions that have the smallest value of one of the four parameters. We will refer to strongly universal nets by the names of the form N_i, and to weakly universal nets by the names of the form N'_i. Note that we do not describe nets N_5 and N'_5 (i.e. the pair N_4 and N'_4 is directly followed by N_6 and N'_6) to harmonize with the nomenclature from [7], which also presents a non-deterministic construction omitted in the present work.

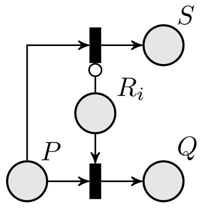
4.1. Minimising the Transition Degree

In this subsection we will show how to construct a strongly and a weakly universal Petri net with inhibitor arcs, with transitions of degree of most 3, based on the universal register machines from Ivan Korec's work [13].

We will mainly rely on the strongly universal U_{22} and the weakly universal U_{20} from [13]. Since these machines only use instructions of type increment and zero-check-and-decrement, it suffices to describe how such operations can be carried out in Petri nets. One of the simplest ideas is representing registers as places and also allocating a place per state. The nets carrying out the two types of instructions we are interested in are shown in Figure 2. The simulation of the increment is straightforward: the token moves from place P into place Q and adds one token to R_i along the way. The zero-check-and-decrement is simulated using an inhibitor arc connected to R_i: if R_i is not empty, the token can only move from P to Q removing a token from R_i along the way, while if R_i is empty, the token can only move to S.



(a) $(p, A(i), q)$



(b) $(p, S(i), q, s)$

Figure 2: Direct simulation of increment and zero-check-and-decrement

We can now directly construct the universal Petri net N_1 simulating the register machine U_{22} by iteratively translating all of its instructions. This Petri net has 22 places for states and 8 places for registers, 30 states all in all. There are 34 transitions in this net, as many as there are arcs in the graph representation of U_{22}. To count the number of inhibitor arcs in N_1, remark that one will be required per each decrement instruction of U_{22}, except for q_{32}, because it suffices to have the net halt with a token in Q_{32} if the place corresponding to register R_4 cannot be decremented. Correspondingly, N_1 has 12 inhibitor arcs. Finally, the maximal transition degree is 3, which means that the size of N_1 is $(30, 34, 12, 3)$.

At the initial marking N_1 will have one token in place Q_1 corresponding to state q_1 of U_{22}, the code of the machine to be simulated in place R_1, and the input value in place R_2. The output of N_1 is to be read from place R_0. Therefore in order to simulate the computation of an arbitrary Petri net N with one input place, N_1 shall be provided with the appropriate coding of N and its input in places R_1 and R_2 respectively. The net N_1 is strongly universal in the sense of the relation $\Phi_N(x) = \Phi^2_{N_1}(g(N), x)$, where Φ_N is the function computed by the Petri net N and g is a function assigning a number to every Petri net, in some fixed enumeration (like Gödel numbering).

The same approach can be applied to simulating the weakly universal register machine U_{20}, yielding the weakly universal Petri net N'_1 of size $(27, 31, 11, 3)$.

Nets N_1 and N'_1 achieve the minimal value of the transition degree necessary for computational completeness. Indeed, a Petri net (even with inhibitor arcs) which only has transitions of degree 2 is bounded; even more, the total count of tokens present in such a net at any time can never increase.

4.2. Minimising the Number of Transitions

Because of the similarity of the semantics of Petri net transitions and the arcs of a generalized register machine, Petri nets can be used to directly simulate the former class of computing devices.

Consider a generalized register machine $M = (R, G, q_0, F, z, nz, add, sub)$, with the underlying multigraph $G = (Q, E, s, t)$. We will construct a Petri net N with the weight function W simulating M. As before, we will represent the states Q and the registers R as places, and, for every edge $e \in E$, with $s(e) = q$ and $t(e) = q'$, we will add a Petri net transition T which will have:

- an arc coming from the state place Q (representing the state q) and an arc going into the state place Q' (representing the state q'):

$$W(Q, T) = 1, \quad W(T, Q') = 1;$$

- an arc going into each of the register places representing the registers incremented by e:

$$W(T, R_i) = a_e(i), \quad \text{for } a_e = add(e), \ i \in R,$$

- an inhibitor arc to each of the register places representing the registers which should be zero in order for e to be enabled:

$$W(R_i, T) = -1, \quad \text{for } i \in z(e),$$

- an arc coming from the register places corresponding to registers decremented by e:

$$W(R_i, T) = 1, \quad \text{for } i \in sub(e),$$

- an arc coming from the register place and an arc going into the same register place for registers which have to be non-zero for e to be enabled, but which are not decremented by e:

$$W(R_i, T) = W(R_i, T) = 1, \quad \text{for } i \in nz(e) \setminus sub(e).$$

Using this approach to simulate the generalized register machine U_7 from Table 1, we obtain the strongly universal Petri net N_2 of size $(14, 23, 30, 6)$. By simulating the generalized register machine U_7', we construct the weakly universal Petri net N_2' of size $(13, 21, 23, 6)$. The initial markings of both Petri nets have one token in place Q_1, which corresponds to the initial state, the code of the simulated register machine in place R_1, and the input value (exponentially coded in the case of N_2') in place R_2. The output of both networks will be found in the register place R_0.

We can further reduce the number of places, while keeping the number of transitions low, by coding the current state number in binary. If the simulated machine has n states, we will use $n_p = \lceil \log_2 n \rceil$ places to encode the current state number in the following way: place Q_i, $0 \leq i < n_p$, will contain a token if the i-th bit of the binary representation of n is 1, and will be empty otherwise. All transitions of such a Petri net will thus depend on all the state places Q_i, $0 \leq i < n_p$, and will produce the new marking of the state places corresponding to the next state number.

Remark that the choice of binary codes for states may influence the total number of inhibitor arcs. Indeed, every transition simulating an arc originating in state q will need to use as many inhibitor arcs as there are zeroes in the binary code assigned to q. Therefore, to keep the number of inhibitor arcs low, we will assign numbers with more non-zero bits to states with more outgoing transitions. This approach yields a strongly universal Petri net N_3 of size $(11, 23, 37, 10)$ and a weakly universal Petri net N_3' of size $(10, 21, 30, 10)$. In the initial marking, the state places of both nets will contain the binary value $(010)_2$, which is the code of the initial states of both U_7 and U_7'.

Of course, one need not restrict oneself to the simulation of fully compressed generalized register machines (i.e. machines which cannot be further compressed). For example, one could consider only compressing the states corresponding to increment instructions; such partial compression has interesting applications to the construction of small universal multiset rewriting systems (e.g. [1, 2]). In the case of Petri nets, this approach allows building a strongly universal Petri net N_4 of size $(21, 25, 12, 5)$ and a weakly universal Petri net N_4' of size $(19, 23, 11, 5)$, which use the same number of transitions as the other two pairs of universal nets shown in this subsection: 23 and 21 for strong and weak universality, respectively.

4.3. Minimising the Number of Places

In this subsection we will construct strongly and weakly universal Petri nets with 5 and 4 places respectively. The constructions will be based on simulations of the universal 3- and 2-register machines described and discussed in Section 2.2.

The work [9] presents a series of non-deterministic constructions attaining a very small number of places by simulating small register machines. It turns out that, even without non-determinism, it is possible to simulate any n-register machine with a Petri net with $n+2$ places only. The idea is to represent state q_i by putting i tokens in state place Q_0 and $|Q|-i$ tokens in place Q_1, and then to have all transitions read the state out of both places Q_0 and Q_1. With this approach, any two multisets $Q_0^i Q_1^{|Q|-i}$ and $Q_0^j Q_1^{|Q|-j}$, representing the markings of state places corresponding to q_i and q_j, $q_i \neq q_j$, are incomparable with respect to the submultiset relation. This means that, if the simulated register machine is deterministic, the resulting Petri net will be deterministic as well.

Figure 3 illustrates how, using such a construction, one can simulate the increment of the register R_k.

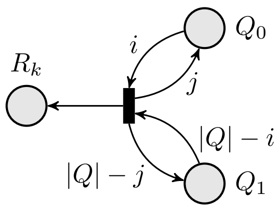


Figure 3: Simulation of $(q_i, A(k), q_j)$ using two state places

We may use this approach to simulate the generalised 3- and 2-register machines described in [1]; this yields a strongly universal Petri net N_6 of size $(5, 246, 123, 378)$ and a weakly universal Petri net N_6' of size $(4, 188, 94, 220)$, which have less transitions and inhibitor arcs than the nets shown under the same names in [7]. We remark that, as different from the model in [1], we avoid collapsing the increments from states q_{30} and q_{31} from Korec's machines U_{22} and U_{20}, because performing the two operations in one single multiplication loop leads to a considerable and undesirable explosion in the maximal transition degree. Still, the transition degree in nets N_6 and N_6' is higher than the corresponding parameter in nets N_5 and N_5' from [7], or even in nets N_6 and N_6' from the same source. (To preserve consistency with the naming of Petri nets in [7], we will not define any nets with index 5 in the present article.)

4.4. Minimising the Number of Inhibitor Arcs

In the previous subsections we saw that it was possible to construct universal Petri nets with as few as four places by having rather complex transitions and by employing an important number of inhibitor arcs. We will now show that it is possible to construct a Petri net which will only have as many inhibitor arcs as there are registers in the simulated register machine. This can be achieved by "outsourcing" the actual zero-check-and-decrement action to special checker subnets instead of using an inhibitor arc per each $S(i)$ instruction. Figure 4 shows how a decrement can be simulated in this way. Essentially, the state token in Q_j is "split" into a token in place C_i activating the checker subnet, and another token which waits in place Q_j' for the result of the checker. The checker subnet is the exact copy of the net from Figure 2 simulating a decrement instruction, and it has the same function.

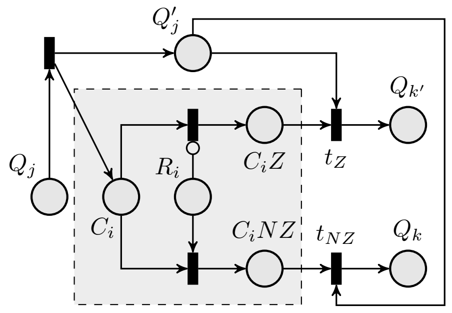


Figure 4: A Petri net simulating a *RiZM* instruction using a checker subnet

We can now use checker subnets to simulate the 3- and 2-register discussed in Section 2.2 and described in detail in [7]. To simulate these machines, we will need, first of all, one place per register, plus three places per checker subnet. To keep the maximal transition degree at 3, we might connect the checker subnets from Figure 4 directly, which would cost us an extra place per decrement instruction. We will prefer avoiding the extra joining and splitting of tokens in places Q_i, which will mean exactly one place per instruction and the maximal transition degree equal to 4. We will need as many transitions as the are increments in the simulated machine, plus double the number of decrements, plus two transition per checker subnet. Finally, we will only need as many inhibitor arcs as there are registers. We can therefore construct the strongly universal net N_7 of size $(379, 513, 3, 4)$ and the weakly universal net N'_7 of size $(285, 392, 2, 4)$.

We remark that, since reachability is decidable for Petri nets with one inhibitor arc [4, 18], net N'_7 uses the minimal number of inhibitor arcs to achieve universality.

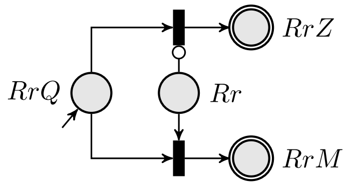
In the construction of N_7 and N'_7 we did not use the register machines from [1], because we represented each instruction separately, while the machines from the cited article have a number of increment instructions which exceeds by far the number of places in N_7 and N'_7. Yet, we may choose to not represent each individual increment, which allows considerably reducing the number of places at the expense of an increase in the maximal transition degree. In this situation, we can use the constructions given in [1] and build the strongly universal Petri net N_8 of size $(135, 252, 3, 136)$ as well as the weakly universal Petri net N'_8 of size $(106, 194, 2, 36)$.

5. Modular Design by Subnets

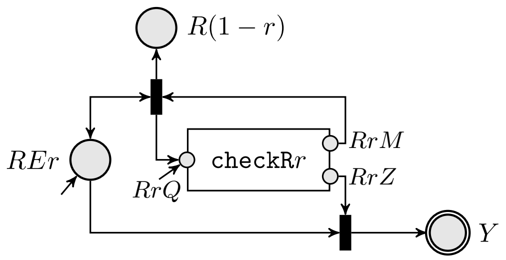
In this section, we will continue exploiting the principle of factoring out functions of subnets with the goal of increasing the reuse of these modules. Instead of building universal 3- and 2-register machines, we will describe Petri nets simulating Korec's U_{22} and U_{20} directly, using Minsky's exponential encoding. The shown Petri nets will be constituted out of building blocks carrying out some elementary operations: zero-check, division, multiplication, as well copying the contents around. To avoid duplicating chains of increments for multiplication and division, the corresponding blocks will be parameterized: the prime number to multiply or divide by will be stored in a place or a group of places accessible to the subnet. Furthermore, the subnets will often have multiple inputs and outputs, as well as sometimes share internal places.

A preliminary remark is due here about representing finite sets of numbers (e.g., the primes of the exponential encoding). Just storing the respective number in a place has several clear disadvantages, one of which is not being able to know which value exactly is stored without using additional structures, like, for example, inhibitor arcs. One obvious candidate for a storage strategy is the complementary encoding, as the one used in Subsection 4.3. Another strategy would be to have one place per member of the set of numbers. Since, the choice of the strategy depends on what size parameter we are optimising for, we will not explicitly define it when introducing subnets. Graphically, the subnets storing members of such sets of numbers will be depicted as dotted squares.

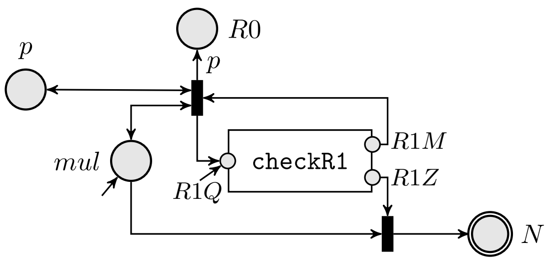
We will now define the subnets, simultaneously assessing their combinatorial complexity. On the pictures, the input places will be pointed at by dangling arrows, while the output places will be contoured in double.



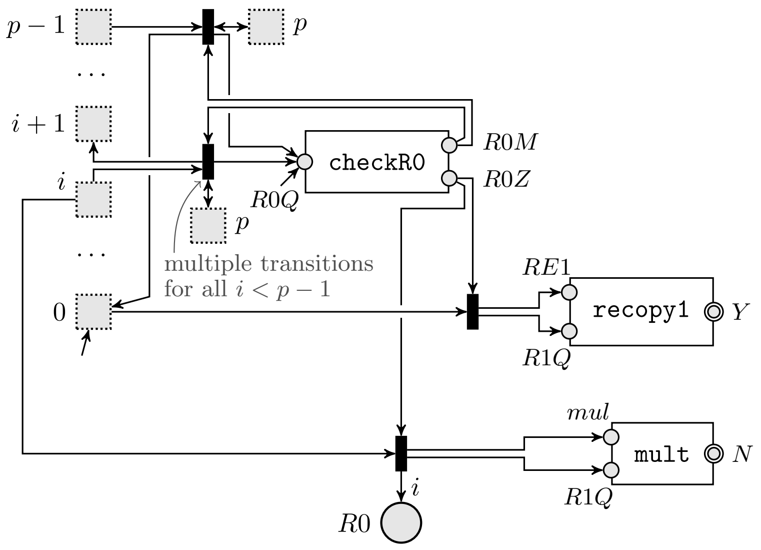
Checker subnet `checkR`r, for every register r. Used to decrement register if possible, final place depends on whether decrement was successful.



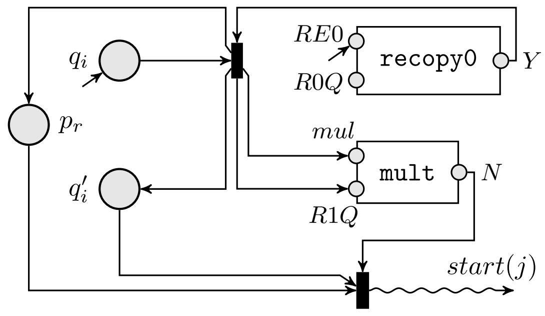
Recopy subnet `recopy`r, for $r \in \{0, 1\}$. Used to move all tokens from Rr to $R(1-r)$.



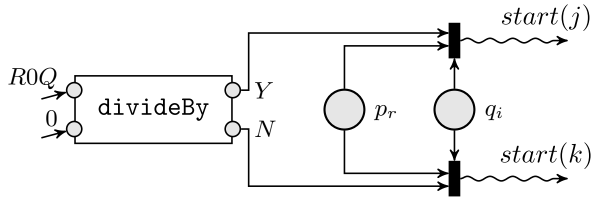
Multiplication subnet `mult`. Used to multiply $R1$ by p, given as a parameter, into $R0$. It is similar to `recopy1`, but output is multiplied by the parameter.



Division subnet `divideBy`. Used to divide $R0$ by p, given as a parameter, into $R1$. If the division produces the remainder, restores the value into $R0$, otherwise recopies the value to $R0$. Final place depends on whether the division was successful. This subnet directly corresponds to the full simulation of a SUB instruction of a register encoded by power of p. Of course, it is enough to represent the position of p in the list of possible values, not the value itself. In order to be able to use only one updating transition per value of i, it suffices to encode p as $A^j B^{m-1-j}$, where j is the zero-based position of p in the list of possible values, and m is the length of this list. in the increasing order. Then, $p > i+1$ can be verified exactly by A^j, where j-th prime is the smallest one exceeding $i+1$. Graphically, the fact that p or i are not directly represented by the number of tokens in a place is illustrated by dotted squares. Note that, for reasons of readability, the picture contains two copies of the dotted place p, which refer to the same place.



ADD subnet `ADD`i. Used for the simulation of the ADD instruction. $start(j) = q_j\ RE0$ if j is an ADD instruction, otherwise $start(j) = q_j\ p_{r_j}\ R0Q\ 0$.



SUB subnet `SUB`i. Used for the simulation of the SUB instruction.

We remark that above subnets implement the blocks of the Minsky algorithm, so it should be clear that their combination allows to construct the simulation of any register machine using a Petri net. The global parameters, like the number of places containing an unbounded number of tokens (corresponding to registers of the simulating machine) and the encoding of the bounded values permit to obtain universal Petri nets minimizing different descriptional complexity parameters.

For example, suppose that we want to implement the simulation of U_{20} using 2 unbounded places (registers). Suppose the optimization priorities are inhibitors, places, transitions, and the maximal transition degree, in order. Then using the subnets above we need following shared places: $R0$, $R0Q$, $R0M$, $R0Z$, $R1$, $R1Q$, $R1M$, $R1Z$, $RE0$, $RE1$, Y, N, mul, A, B, C, D, and 20 places q_i and 9 places q'_i, i.e., **46** places in total. The number of transitions by subnets is equal to 81. The construction only uses **2** inhibitors. The maximal diameter is 40, so we have a weakly universal Petri N'_9 net with descriptional complexity $(46, 81, 2, 40)$. In a similar way, based on U_{22}, the Petri net N_9 of size $(51, 89, 3, 45)$ can be obtained.

6. Conclusion

In this article we considered the question of universality for Petri nets with inhibitor arcs and gave several small universal Petri nets of different descriptional complexity. The size of the constructed strongly and weakly universal nets are given in Tables 2 and 3, respectively. We remind that, in this paper, we skip the index 5 in the names of Petri nets to harmonize with the nomenclature from [7].

Table 2: Strongly universal Petri nets

	N_1	N_2	N_3	N_4	N_6	N_7	N_8	N_9
Places	30	14	11	21	5	379	135	51
Transitions	34	23	23	25	246	513	252	89
Inhibitor arcs	12	30	37	12	123	3	3	3
Maximal degree	3	6	10	5	378	4	136	45

Nets N_1 and N_1' were obtained by directly simulating Korec's register machines. Nets with indices 2 through 4 simulate universal generalised register machines at different degrees of state compression, with and without binary coding of states. Nets N_6 and N_6' were built by simulating universal 3- and 2-register machines, by storing state numbers in complementary coding in two places. Nets with indices 7 and 8 rely on checker subnets, while nets N_9 and N_9' take the idea of subnets even further and rely heavily on atomic blocks carrying out elementary operations.

We remark that nets with indices 1 through 4, as well as 7 and 8, simulate the corresponding register machines in real time — one evolution step of the Petri net is required to simulate an instruction of the register machine. Other nets simulate corresponding register machines with exponential slowdown. Since Korec's machines simulate any other machine with exponential slowdown already, and since register machines simulate Petri nets with a constant slowdown, nets with indices 1 through 4, 7, and 8 can simulate any other Petri net with exponential slowdown, while the other nets carry out the simulations with doubly exponential slowdown. This brings out the trade-off between certain size parameters and the simulation speed.

Some of our constructions achieve the theoretical minimum for the corresponding parameters of descriptional complexity. Thus, nets N_1 and N_1' attain the minimal possible value for transition degree: 3, while weakly universal nets with indices 7 through 9 achieve universality with the smallest possible number of inhibitor arcs: 2. As for the number of places and transitions, we conjecture that the values that we give for these parameters: 23 and 21 transitions, and 5 and 4 places for strong and weak universality respectively, cannot be significantly improved upon because of inherent limitations of Petri nets with inhibitor arcs.

Our constructions bring out a number of interesting trade-offs. Comparing nets N_1 and N_1' from with N_2 and N_2', N_3 and N_3', and N_4 and N_4', we remark that a reduction of the number of places leads to an increase in the number of inhibitor arcs and the maximal degree of a transition. Nets N_6, and N_6' accentuate this trade-off even more: they are universal with 5 and 4 places, but rely on considerably more inhibitor arcs and transitions of big degrees.

Table 3: Weakly universal Petri nets

	N_1'	N_2'	N_3'	N_4'	N_6'	N_7'	N_8'	N_9'
Places	27	13	10	19	4	285	106	46
Transitions	31	21	21	23	188	392	194	81
Inhibitor arcs	11	23	30	11	94	2	2	2
Maximal degree	3	6	10	5	220	4	36	40

Finally, nets N_9 and N_9' show how factoring out certain functions allows drastically reducing all the components of the size tuple of a Petri net — instead of replicating subnets with minor variations, we built nets N_9 and N_9' out of modules which are heavily reused. Another feature of these two nets is that they carry out exponential encoding directly, instead of simulating a 3- or 2-register machines. This approach seems likely to have applications to further reducing of size of universal Petri nets.

References

[1] A. Alhazov, R. Freund, Variants of Small Universal P Systems with Catalysts. *Fundamenta Informaticae* **138** (2015) 1-2, 227–250.

[2] A. Alhazov, S. Verlan, Minimization Strategies for Maximally Parallel Multiset Rewriting Systems. *Theoretical Computer Science* **412** (2011) 17, 1581 – 1591.

[3] I. M. Barzdin, Ob odnom klasse machin Turinga (machiny Minskogo), Russian. *Algebra i Logika* **1** (1963), 42–51.

[4] H. K. Büning, T. Lettmann, E. W. Mayr, Projections of Vector Addition System Reachability Sets Are Semilinear. *Theoretical Computer Science* **64** (1989) 3, 343–350.

[5] R. Freund, O. H. Ibarra, G. Păun, H.-C. Yen, Matrix Languages, Register Machines, Vector Addition Systems. In: M. A. G. Naranjo, A. Riscos-Nez, F. J. Romero-Campero, D. Sburlan (eds.), *Proceedings of the Third Brainstorming Week on Membrane Computing*. University of Sevilla, 2005, 155–168.

[6] R. Freund, M. Oswald, A Small Universal Antiport P System with Forbidden Context. In: H. Leung, G. Pighizzini (eds.), *8th International Workshop on Descriptional Complexity of Formal Systems - DCFS 2006, Las Cruces, New Mexico, USA, June 21 - 23, 2006. Proceedings*. New Mexico State University, Las Cruces, New Mexico, USA, 2006, 259–266.

[7] S. Ivanov, *On the Power and Universality of Biologically-inspired Models of Computation*. Ph.D. thesis, Université Paris Est, Paris, 2015.

[8] S. Ivanov, E. Pelz, S. Verlan, Small Universal Petri Nets with Inhibitor Arcs. *CoRR* **abs/1312.4414** (2013).

[9] S. Ivanov, E. Pelz, S. Verlan, Small Universal Non-deterministic Petri Nets with Inhibitor Arcs. In: H. Jürgensen, J. Karhumäki, A. Okhotin (eds.), *Descriptional Complexity of Formal Systems - 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*. Lecture Notes in Computer Science 8614, Springer, 2014, 186–197.

[10] S. Ivanov, E. Pelz, S. Verlan, Small Universal Petri Nets with Inhibitor Arcs. In: *Computability in Europe*. 2014.

[11] H. Jürgensen, J. Karhumäki, A. Okhotin (eds.), *Descriptional Complexity of Formal Systems - 16th International Workshop, DCFS 2014, Turku, Finland, August 5-8, 2014. Proceedings*. Lecture Notes in Computer Science 8614, Springer, 2014.

[12] P. Koiran, C. Moore, Closed-form Analytic Maps in One and Two Dimensions Can Simulate Universal Turing Machines. *Theoretical Computer Science* **210** (1999) 1, 217–223.

[13] I. Korec, Small Universal Register Machines. *Theoretical Computer Science* **168** (1996) 2, 267–301.

[14] M. Minsky, Size and structure of universal Turing machines using tag systems. In: *Recursive Function Theory: Proceedings, Symposium in Pure Mathematics, Provelence*. 5, 1962, 229–238.

[15] M. Minsky, *Computations: Finite and Infinite Machines*. Prentice Hall, Englewood Cliffts, NJ, 1967.

[16] T. Neary, D. Woods, The Complexity of Small Universal Turing Machines: A Survey. In: M. Bieliková, G. Friedrich, G. Gottlob, S. Katzenbeisser, G. Turán (eds.), *SOFSEM 2012: 38th Conference on Current Trends in Theory and Practice of Computer Science*. Lecture Notes in Computer Science 7147, Springer, 2012, 385–405.

[17] E. Pelz, Closure Properties of Deterministic Petri nets. In: *Symposium on Theoretical Aspects of Computer Science, STACS '87*. Lecture Notes in Computer Science 247, Springer, 1986, 371–382.

[18] K. Reinhardt, Reachability in Petri Nets with Inhibitor Arcs. *Electronic Notes in Theoretical Computer Science* **223** (2008), 239–264.

[19] Y. Rogozhin, Small Universal Turing Machines. *Theoretical Computer Science* **168** (1996) 2, 215–240.

[20] R. Schroeppel, A Two Counter Machine Cannot Calculate 2N. In: *AI Memos*. MIT AI Lab, 1972.

[21] C. E. Shannon, A Universal Turing Machine with Two Internal States. *Automata Studies, Annals of Mathematics Studies* **34** (1956), 157–165.

[22] A. M. Turing, On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* **42** (1936) 2, 230–265.

[23] S. Watanabe, 5-symbol 8-state and 5-symbol 6-state universal Turing machines. *Journal of the ACM* **8** (1961) 4, 476–483.

[24] D. Woods, T. Neary, The Complexity of Small Universal Turing Machines: A Survey. *Theoretical Computer Science* **410** (2009) 4-5, 443–450.

[25] D. A. Zaitsev, Universal Petri Net. *Cybernetics and Systems Analysis* **48** (2012) 4, 498–511.

[26] D. A. Zaitsev, A Small Universal Petri Net. *EPTCS* **128** (2013), 190–202. In Proceedings of Machines, Computations and Universality (MCU 2013), arXiv:1309.1043.