Small Universal Non-deterministic Petri Nets with Inhibitor Arcs

Sergiu Ivanov, Elisabeth Pelz, and Sergey Verlan

Laboratoire d'Algorithmique, Complexité et Logique, Université Paris Est
61, av. du gén. de Gaulle, 94010 Créteil, France
{sergiu.ivanov,pelz,verlan}@u-pec.fr

Abstract. This paper investigates the universality problem for Petri nets with inhibitor arcs. Four descriptional complexity parameters are considered: the number of places, transitions, inhibitor arcs, and the maximal degree of a transition. Each of these parameters is aimed to be minimized, a special attention being given to the number of places. Four constructions are presented having the following values of parameters (listed in the above order): $(5, 877, 1022, 729)$, $(5, 1024, 1316, 379)$, $(4, 668, 778, 555)$, and $(4, 780, 1002, 299)$. The decrease of the number of places with respect to previous work is primarily due to the consideration of non-deterministic computations in Petri nets. Using equivalencies between models our results can be translated to multiset rewriting with forbidding conditions, or to P systems with inhibitors.

1 Introduction

The research of small universal computing devices is an amazing and continuous research topic since many decades. It started by A. Turing proposal of an universal (Turing) machine [17] capable of simulating the computation of any other (Turing) machine. This universal machine takes as input a description of the machine to simulate, the contents of its input tape, and computes the result of its execution on the given input.

More generally, the universality problem for a class of computing devices (or functions) $\mathfrak{C}$ consists in finding a fixed element $\mathcal{M}$ of $\mathfrak{C}$ able to simulate the computation of any element $\mathcal{M}'$ of $\mathfrak{C}$ using an appropriate fixed encoding. More precisely, if $\mathcal{M}'$ computes y on an input x (we will write this as $\mathcal{M}'(x) = y$), then $\mathcal{M}'(x) = f(\mathcal{M}(g(\mathcal{M}'), h(x)))$, where h and f are the encoding and decoding functions, respectively, and g is the function retrieving the number of $\mathcal{M}'$ in some fixed enumeration of $\mathfrak{C}$. Although general recursive functions may be used for encoding and decoding, we would prefer to see the computing device, and not the encoders and decoders, do most of the work. Hence we prefer computationally very simple encoding and decoding functions [18], like the typically used $f(x) = \log_2(x)$ and $h(x) = 2^x$.

In what follows, we will keep to the terminology considered by Korec [6] and call the element $\mathcal{M}$ (weakly) universal for $\mathfrak{C}$. We shall call $\mathcal{M}$ strongly universal (for $\mathfrak{C}$) if the encoding and decoding functions are identities.

H. Jürgensen et al. (Eds.): DCFS 2014, LNCS 8614, pp. 186–197, 2014.

Some authors [7,6] implicitly consider only the strong notion of universality as the encoding and decoding functions can perform quite complicated transformations, which are not necessarily doable in the original devices. For example, Minsky's proof of (weak) universality of register machines with two counters [11] makes use of exponential (resp. logarithmic) encoding (resp. decoding) functions, while it is known that such functions cannot be computed directly (without encoding) on these machines [2,16]. We refer to [6] for a detailed discussion of different variants of the universality and to [8] for a survey on this topic. Generally, the class of all partially recursive functions is considered as $\mathfrak{C}$, but it is possible to have a narrower class, e.g. the class of all primitive recursive functions, which is known to admit a universal generally recursive function [7]. We remark that in the case of devices not working with integers directly, some natural coding of integers should be used in order to consider the above notions.

Small universal devices have mostly theoretical importance as they demonstrate the minimal ingredients needed to achieve a complex (universal) computation. Their construction is a long-standing and fascinating challenge involving a lot of interconnections between different models, constructions, and encodings.

In [1] a small universal maximally parallel multiset rewriting system is constructed. Due to equivalences between Petri nets and multiset rewriting systems, this result can be seen as a universal Petri net working with max step semantics. For the traditional class of Petri nets there were no known universality constructions for a long time. Recently, Zaitsev has investigated the universality of Petri nets with inhibitor arcs and priorities [20] and has constructed a small universal net with 14 places and 29 transitions (for nets without priorities the same author obtained a universal net with 500 places and 500 transitions [19]). We remark that inhibitor arcs and priorities are equivalent extensions for Petri nets in terms of computational power, so using both concepts together is not necessary for universality constructions.

In [4], a series of small strongly and weakly universal Petri nets was constructed and compared to Zaitsev' results. The cited paper only focuses on Petri nets with deterministic evolution, however. The present article considers a natural question of whether allowing non-deterministic evolution may be exploited to further minimize certain parameters. We show some trade-offs: trying to reduce the number of places of a universal Petri net seems to imply an increase in the other parameters, e.g., the maximal transition degree. We also show how linear programming could be used to minimize this degree while keeping the number of places small.

Quite importantly, due to the equivalence between Petri nets, multiset rewriting, and asynchronous P systems [13], the results from this paper can be immediately translated to corresponding universality statements for these models.

2 Preliminaries

In this section we only recall some basic notions and notations; see [15] for further details. An alphabet is a finite non-empty set of symbols. Given an alphabet V, we designate by V^* the set of all strings over V, including the empty string,

λ. For each $x \in V^*$ and $a \in V$, $|x|_a$ denotes the number of occurrences of the symbol a in x. A finite multiset over V is a mapping $X : V \longrightarrow \mathbb{N}$, where $\mathbb{N}$ denotes the set of non-negative integers. $X(a)$ is said to be the multiplicity of a in X.

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 (called also rules) of the following form:

1. *(Increment)* $(p, RiP, q) \in P$, $p, q \in Q, p \neq q, R_i \in R$ (being in state p, increment register R_i and go to state q).
2. *(Decrement)* $(p, RiM, q) \in P$, $p, q \in Q, p \neq q, R_i \in R$ (being in state p, decrement register R_i and go to state q).
3. *(Zero check)* $(p, Ri, q, s) \in P$, $p, q, s \in Q, R_i \in R$ (being in state p, go to q if register R_i is not zero or to s otherwise).
4. *(Zero test and decrement)* $(p, RiZM, q, s) \in P$, $p, q, s \in Q, R_i \in R$ (being in state p, decrement register R_i and go to q if successful or to s otherwise).
5. *(Stop)* $(q_f, STOP)$ (may be associated only to the final state q_f).

Note that a $RiZM$ instruction can be used to simulate both a RiM instruction and a Ri instruction.

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 registers. A transition of the register machine consists in updating/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 $(q_f, y, 0, \ldots, 0)$.

It is well-known that register machines compute all partial recursive functions and only them [11]. Therefore, every register machine M with n registers can be associated with the function it computes: an m-ary partial recursive function Φ_M^m, where $m \leq n$. Let $\Phi_0, \Phi_1, \Phi_2, \ldots$, be a fixed enumeration of the set of unary partial recursive functions. Then, a register machine M is said to be *strongly universal* [6] if there exists a recursive function g such that $\Phi_x(y) = \Phi_M^2(g(x), y)$ holds for all $x, y \in \mathbb{N}$. A register machine M is said to be *(weakly) universal* if there exist recursive functions f, g, h such that $\Phi_x(y) = f(\Phi_M^2(g(x), h(y)))$ holds for all $x, y \in \mathbb{N}$. We remark that here the meaning of the term weakly universal is different from the Turing machines case, where it is commonly used to denote a universal machine working on a tape that has an infinite initial configuration with one constant word repeated to the right, of and another to the left of the input [9].

2.2 Petri Nets

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

PT-nets are usually represented by diagrams where places are drawn as circles, transitions are drawn as squares annotated with their location, and a directed arc (x, y) is added between x and y if $W(x, y) \geq 1$. These arcs are then annotated with their weight if this one is 2 or more. Arcs having the weight -1 are called *inhibitor* arcs and are drawn such that the arcs end 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:

$$degree(t) = \sum_{p \in P} \big|W(p, t)\big| + \big|W(t, p)\big|.$$

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

Given a PT-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 $p \in P$, for which $W(p, t) \geq 0$, $pre_{N(t)}(p) = W(p, t)$ and $post_{N(t)}(p) = W(t, p)$. A state of N, which is called a *marking*, is a multiset M over P; in particular, for every $p \in P$, $M(p)$ represents the number of *tokens* present 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. An enabled transition t 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 $p \in P$, the firing transition t consumes $pre_{N(t)}(p)$ tokens and produces $post_{N(t)}(p)$ tokens). We denote this as $M \xrightarrow{t} M'$.

For the purposes of this paper, we have to define which kind of PT-nets can execute computations (e.g. compute partially recursive functions). In such a net some distinguished places $i_1, \ldots, i_k$, $k > 0$ from P are called *input* places (which are normally different from the places marked in M_0 containing the control tokens) and one other, $i_0 \in P$, is 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 in some marking M_f, i.e. in M_f no transition is enabled. Thus we have $M'_0 \xrightarrow{*} M_f$ and there are no M'_f and $t \in T$ 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 could compute a set of numbers.

If, for any reachable marking M, 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 [14]. Otherwise the Petri net is called non-deterministic, and it is this kind of nets that we will focus on in this paper.

The *size* of a Petri net is 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 of the Petri net provide the fundamental information about its structure and can be further used to reason about its other features (e.g., the average number of inhibitor arcs per transition). Moreover, each of these parameters 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. Remember that, when counting the degree of a transition, we take into account the weights of the arcs it involves, even though the degree is often considered to be the sum of the number of input and output places of the transition.

3 Universal Register Machines with Few Registers

A rather well-known result on the computational power of register machines is that there exists a strongly universal machine with 3 registers and a weakly universal machine with 2 registers only [10]. Since it seems natural to represent registers as places [4], we have special interest in machines with a small number of registers, and, for the purposes of this paper, we have actually constructed a strongly universal 3-register machine U_3 and a weakly universal 2-register machine U_2 following the ideas proposed in [10].

Roughly, the construction is based on exponential coding of the configuration of an arbitrary register machine and simulating increments and decrements as multiplications and divisions. Two registers are enough for these purposes, and a third one is required for exponentiation of the input and computing of the logarithm to retrieve the output. To construct the weakly universal U_2, we simulate the 20-state 8-register weakly universal register machine U_{20} described in [6]. The 2-register machine obtained in this way has 112 decrement and 165 increment instructions: 278 states all in all (including a final state). Simulating the 22-state 8-register machine U_{22} described in the same paper, and further adding exponentiation and logarithm, allows building the strongly universal 3-register machine U_3 with 147 decrement and 217 increment instructions: 365 states in total.

Clearly, U_2 and U_3 simulate the corresponding machines from [6] with exponential slowdown. However, the machines by Korec simulate partial recursive functions with exponential slowdown, too (cf. Theorem 4.2 in [6]). This means that the slowdown of the machines U_2 and U_3 with respect to the (indirectly) simulated partial recursive functions is doubly exponential.

Both register machines use the register R_0 to store the exponentially-coded values of the simulated register machine, and the 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 produces its output in the third register, R_2.

An important remark with regard to the strong universality of U_3 is due here: since we use one register for input, we are only able to directly simulate unary partial recursive functions. Nevertheless, Section 9 of [6] 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 pretty naturally adapted to the register machine U_2 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. Such register machines can be translated into Petri nets by the same techniques as the ones we will show for U_2 and U_3 in the coming sections.

Section 2 of [5] describes the construction of a universal three-register machine with 130 states. However, in that 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 the corresponding construction in terms of elementary register machine commands as we use in U_2 and U_3, would yield more than 450 instructions.

4 Non-deterministic Simulation of Register Machines by Petri Nets

In this section we will show that a register machine with n registers can be simulated by a non-deterministic Petri net with only $n + 2$ places. We recall that in the non-deterministic semantics, we only consider those branches of computation which halt. The basic idea is representing a state number of a register machine in unary encoding as the number of tokens in a (single) place of the Petri net, and then using another place to assure that the transformations of state numbers happen correctly.

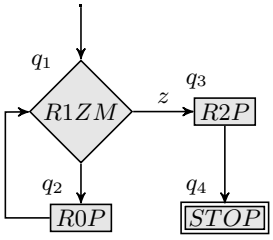

Fig. 1. The toy register machine S

Consider the following register machine $S = (Q, R, q_1, q_4, P)$, where $Q = \{q_1, q_2, q_3, q_4\}$, $R = \{R_0, R_1, R_2\}$ and the set of instructions P is defined as $P = \{(q_1, R_1M, q_2, q_3), (q_2, R0P, q_1), (q_3, R2P, q_4), (q_4, STOP)\}$. This machine is depicted on Figure 1 using a standard flow-chart notation.

This machine adds the contents of the register r_1 to the register r_0, eventually sets r_1 to zero, and increments r_2 once. The corresponding non-deterministic Petri net N is shown in Figure 2. This Petri net uses a place to store the value of each register, and two more places to store state numbers and validate state transitions. Note that we do not need to represent the final state q_4 in the Petri net. It suffices that N carries out the operation associated with q_3 and halts.

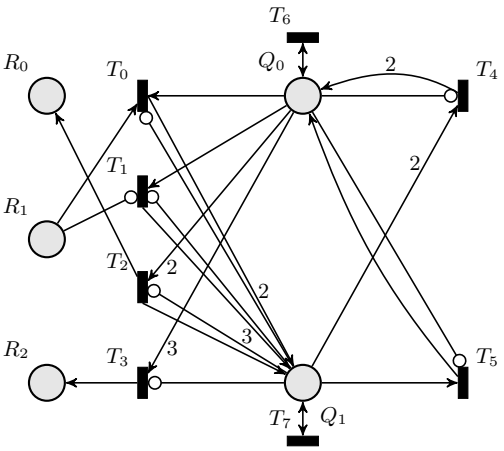

Fig. 2. The non-deterministic Petri net N simulating S

The Petri net N starts with one token in the state place Q_0, which corresponds to the number of the initial state q_1 of S. The simulation of an arc in the graph of the register machine S is carried out in two phases. In the first phase, the current state is read from the place Q_0, the corresponding registers are checked and/or modified, and the number of the next state is put into the state place Q_1. The first phase of the simulation is carried out by the transitions T_0 through T_3. The goal of the second phase is checking that state places are always read completely, i.e., that every transition which fires consumes all the tokens from Q_0 or Q_1. The second phase corresponds to the firing of one of the transitions T_4 or T_5.

Transitions T_0 and T_1 simulate state q_1 of S in the following way. If R_1 is nonempty, the transition T_0 can fire, consuming one token from the place R_1 and putting two tokens into Q_1, which corresponds to moving into state q_2. If, however, R_1 is empty, transition T_1 fires and places three tokens into Q_1, which corresponds to moving into state q_3. Similarly, transition T_2 simulates the behavior of S in state q_2, and transition T_3 corresponds to the state q_3. Note that T_3 only empties Q_0 and adds nothing to Q_1, moreover, Q_1 should be empty in order for T_3 to fire. Therefore, after T_3 fires, no more transitions can fire and N halts.

The goal of transitions T_4 and T_5 is moving the number of the next state to simulate from Q_1 to Q_0. At the same time, the two transitions verify that the

correct instruction has just been carried out. Indeed, suppose that Q_0 contains two tokens and R_1 is empty. Then both transition T_0 and T_2 can fire. However, if T_0 fires, it will place two tokens into Q_1, and one token will still be left in Q_0. In this case transitions T_0 through T_5 will be blocked, and the only enabled transitions T_6 and T_7 will make the net loop forever. Similarly, if transition T_5 fires when Q_1 contains more than one token, N will never halt. Therefore, in a halting computation of N, either the place Q_0 or Q_1 contains tokens, but not both at the same time, which assures the correct simulation of the register machine S.

We remark that the number of places in the above simulation does not depend on the number of states of the simulated machine. Hence, it should be rather clear that the same reasoning can be repeated for the 3-register universal machine U_3 to obtain a 5-place strongly universal non-deterministic Petri net N_1. It will have 511 transitions simulating the activity of U_3, 364 more transitions moving the code of the next state from Q_1 to Q_0, and two loops; 877 transitions all in all. It will also have 1022 inhibitor arcs and the maximal transition degree will be 729. In an analogous fashion we can build a 4-place weakly universal Petri net N_2 with 668 transitions of maximal degree 555 and with 778 inhibitor arcs.

5 Decreasing the Transition Degree

In this section we will show that it is possible to almost halve the maximal transition degree of the transitions in the strongly universal non-deterministic Petri net N_1 at the cost of a slight increase in the number of transitions.

First of all, remark that the maximal transition degree in the Petri net N_1 is largely determined by the transitions moving tokens from Q_1 to Q_0: since U_3 contains 365 states including a final one, N_1 must have a transition moving 364 tokens from Q_1 to Q_0, which, together with the inhibitor arc coming from Q_0, results in degree 729. Remember that we try to be precise from the graph-theoretic point of view and count the weights of the arcs as well, while in many Petri-net-related works the degree of such transitions would be considered 3, as they have only one input and two output places.

On the other hand, the transitions actually simulating the activity of U_3 need not be this large: it is possible to code states in such a way that these transitions have an input degree which is much smaller than the output degree or vice versa. The idea therefore is as follows: mirror the transitions which simulate the activity of U_3, so that the simulation happens both when tokens are moved from Q_0 to Q_1 *and* when tokens are moved from Q_1 to Q_0. In the case of our toy register machine S, this will result in the Petri net shown in Figure 3. Observe that the transition T_i' simulates the same instruction of S as T_i does.

In the case of this net, maximal transition degree is determined solely by the way in which the states are represented. Namely, consider a transition T which corresponds to the move from state q_i to state q_j in the simulated register machine. The way in which states are encoded will be given by the function $c : Q \to \mathbb{N}$, where Q is the set of states. Then, the degree of T is $c(q_i) +$

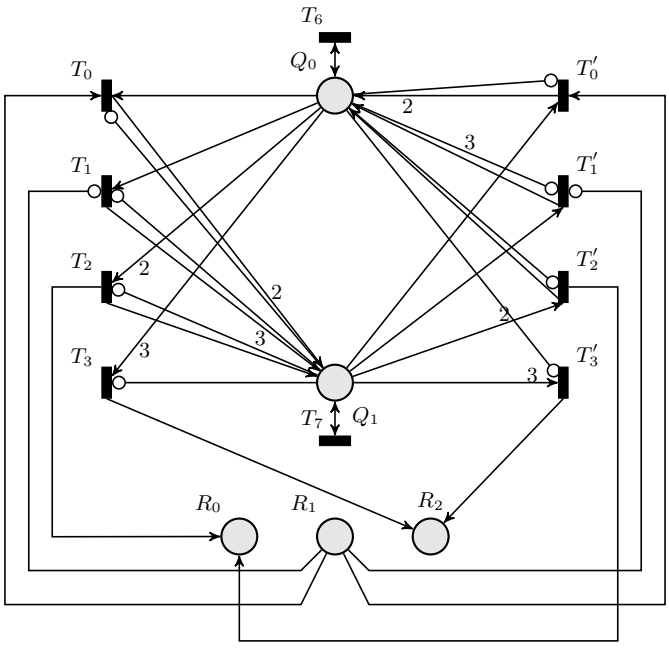

Fig. 3. A mirrored non-deterministic Petri net simulating S

$c(q_j) + 2$, where we add 2 because all transitions read and/or modify a register and are inhibited by either Q_0 or Q_1. We can now define the mapping c via a minimization problem, keeping in mind the goal to minimize the worst transition degree.

We start by remarking that c takes values in the interval 1 through $|Q| - 1$ (we do not need to code the final state). In the optimization problem, we use the following family of variables:

$$c_{i,i'} = \begin{cases} 1, & \text{if } c(q_i) = i', \\ 0, & \text{otherwise,} \end{cases}$$

with the following normalization conditions:

$$\forall q_i \in Q : \sum_{1 \le i' \le |Q|-1} c_{i,i'} = 1 \text{ and } \forall 1 \le i' \le |Q| - 1 : \sum_{q_i \in Q} c_{i,i'} = 1,$$

which require that every state have exactly one code and that each code be picked exactly once.

The family of variables $c_{i,i'}$ can be used to express the cost $c(q_i) + c(q_j)$ for a transition from state q_i to q_j in the following way:

$$c(q_i) + c(q_j) = \sum_{1 \le i' \le |Q|-1} i' \cdot c_{i,i'} + \sum_{1 \le j' \le |Q|-1} j' \cdot c_{j,j'}.$$

For convenience, we will also define the set of pairs of states between which there exists an arc in the graph of the register machine:

$$B = \big\{(q_i, q_j) \in Q \times Q \mid (q_i, RkP, q_j) \in P, \\ \text{or } (q_i, RkP, q_j, s) \in P, \text{ or } (q_i, RkP, s, q_j) \in P\big\}.$$

We can now write the linear programming problem optimizing the cost of the "worst" transition:

$$\text{Minimize } C$$

$$\text{Subject to } \sum_{1 \le i' \le |Q|-1} i' \cdot c_{i,i'} + \sum_{1 \le j' \le |Q|-1} j' \cdot c_{j,j'} \le C, (q_i, q_j) \in B,$$

$$\forall q_i \in Q : \sum_{1 \le i' \le |Q|-1} c_{i,i'} = 1,$$

$$\forall 1 \le i' \le |Q| - 1 : \sum_{q_i \in Q} c_{i,i'} = 1.$$

To attack the instances of this linear programming for U_3 and U_2, we used the Gurobi Optimizer [12]. The problem itself was formulated in the AMPL model description language [3].

Remark that the number of variables in this linear programming problem is $|Q|^2 + 1$, which results in rather large linear programming models in the case of the universal 3- and 2-register machines. U_3, for example, has 364 non-final states, which means 132 496 variables. Furthermore, the first line in the definition of the linear programming problem introduces 511 constraints, each constraint being a linear combination of 729 variables. Finally, the last two lines introduce 728 more constraints involving 364 variables each. This amounts to 1239 constraints involving either 364 or 729 variables.

Due to limited computing resources, we were not able to find the optimal solutions for U_3 and U_2 until now. Nevertheless, we managed to obtain some reasonably good solutions which allow reducing the maximal transition degree to 379 in the case of U_3 and to 277 for U_2. Thus, we obtained a 5-place strongly universal non-deterministic Petri net N_3 with 1024 transitions of maximal degree 379 and 1316 inhibitor arcs, and a 4-place weakly universal non-deterministic Petri net N_4 with 780 transitions of maximal degree 299 and 1002 inhibitor arcs.

6 Main Results

To summarize the results concerned with strong universality, we formulate the following statement.

Theorem 1. *There exist* strongly *universal non-deterministic Petri nets of sizes* $(5, 877, 1022, 729)$ *and* $(5, 1024, 1316, 379)$.

Similarly we can state the following with respect to weakly universal non-deterministic Petri nets:

Theorem 2. *There exist* weakly *universal non-deterministic Petri nets of sizes* $(4, 668, 778, 555)$ *and* $(4, 780, 1002, 299)$.

7 Conclusion

In this paper we constructed two strongly universal and two weakly universal non-deterministic Petri nets. We have shown that dropping the restriction of a deterministic evolution allows a dramatic minimization of the number of places, but produces an important increase in the values of the other parameters.

We remark that, while the strong universality results obtained in this paper implicitly suppose that corresponding Petri nets have a single input (thus computing unary functions), it is possible to generalize them to an n-ary input using the ideas from [6]. Corresponding nets will have $n+2$ places and a linear increase in the number of inhibitors and maximal transition degree.

While our main goal was to minimize the number of places, we did also show that a trade-off still existed between the maximal transition degree on the one side and the number of transitions and inhibitor arcs on the other. It could be interesting to look for other similar trade-offs.

It might be possible to achieve strong universality with 4 places, because the input/output register of U_3 is only modified during the initial and final phases of execution of the machine, when input is read and output is produced [10]. We conjecture that 3 places or less are insufficient to achieve strong universality, however. On the other hand, it may be possible to reduce the values of some other parameters, while keeping the number of states at 5.

In Section 3, we have cited a universal 3-register machine from [5] and said that, if only elementary increments and decrements with zero check are allowed as instructions, this machine would have more instruction than U_3. However, compound instructions do map naturally on Petri net transitions: multiple increments can be carried out in one step by an arc with multiplicity greater than 1. Therefore, the register machine from [5] could be simulated directly, thus reducing the number of transitions, and of inhibitor arcs, but also increasing the maximal transition degree. Constructing the corresponding Petri net might be a worthwhile task that could potentially lead to a better maximal degree measure.

The universality results shown in this paper indirectly rely on exponential coding, which imposes operations of considerable time complexity. A different approach could be used to reduce simulation time, but this would almost certainly result in an increase in the size of the Petri nets.

Finally, we would like to stress that the results we give in this paper can be straightforwardly translated to the domain of P systems [13] and, more generally, multiset rewriting [1].

References

1. Alhazov, A., Verlan, S.: Minimization strategies for maximally parallel multiset rewriting systems. Theoretical Computer Science 412(17), 1581–1591 (2011)
2. Barzdin, I.M.: Ob odnom klasse machin Turinga (machiny Minskogo), russian. Algebra i Logika 1, 42–51 (1963)
3. Fourer, R., Gay, D.M., Kernighan, B.W.: AMPL: A Modeling Language for Mathematical Programming, 2nd edn. Duxbury Press, Brooks/Cole Publishing Company (2002)

4. Ivanov, S., Pelz, E., Verlan, S.: Small universal Petri nets with inhibitor arcs. In: Computability in Europe (2014)
5. Koiran, P., Moore, C.: Closed-form analytic maps in one and two dimensions can simulate universal turing machines. Theor. Comput. Sci. 210(1), 217–223 (1999)
6. Korec, I.: Small universal register machines. Theoretical Computer Science 168(2), 267–301 (1996)
7. Malcev, A.I.: Algorithms and Recursive Functions. Wolters-Noordhoff Pub. Co., Groningen (1970)
8. Margenstern, M.: Frontier between decidability and undecidability: A survey. Theoretical Computer Science 231(2), 217–251 (2000)
9. Margenstern, M.: An algorithm for buiding inrinsically universal automata in hyperbolic spaces. In: Arabnia, H.R., Murgin, M. (eds.) FCS, pp. 3–9. CSREA Press (2006)
10. Minsky, M.: Size and structure of universal Turing machines using tag systems. In: Recursive Function Theory: Proceedings, Symposium in Pure Mathematics, Provelence, vol. 5, pp. 229–238 (1962)
11. Minsky, M.: Computations: Finite and Infinite Machines. Prentice Hall, Englewood Cliffts (1967)
12. Gurobi Optimization, Inc. Gurobi optimizer reference manual (2014)
13. Păun, G.: Membrane Computing. An Introduction. Springer, Berlin (2002)
14. Pelz, E.: Closure properties of deterministic Petri nets. In: Brandenburg, F.J., Wirsing, M., Vidal-Naquet, G. (eds.) STACS 1987. LNCS, vol. 247, pp. 371–382. Springer, Heidelberg (1987)
15. Rozenberg, G., Salomaa, A. (eds.): Handbook of Formal Languages, vol. 1–3. Springer (1997)
16. Schroeppel, R.: A two counter machine cannot calculate 2N. In: AI Memos. MIT AI Lab (1972)
17. Turing, A.M.: On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society 42(2), 230–265 (1936)
18. Woods, D., Neary, T.: The complexity of small universal Turing machines: A survey. Theor. Comput. Sci. 410(4-5), 443–450 (2009)
19. Zaitsev, D.A.: Universal Petri net. Cybernetics and Systems Analysis 48(4), 498–511 (2012)
20. Zaitsev, D.A.: A small universal Petri net. EPTCS 128, 190–202 (2013); In Proceedings of Machines, Computations and Universality (MCU 2013), arXiv:1309.1043