Neural, Parallel, and Scientific Computations 24 (2016) 29-44

COMPUTABILITY POWER OF AN EXTENDED PETRI NET MODEL (APN)

ZVI RETCHKIMAN KONIGSBERG

Centro de Investigacion en Computacion, Instituto Politecnico Nacional,
Mexico D.F, MEXICO

ABSTRACT. The notion of computability has its origin in the works of the logicians Church, Gödel, Kleene, Post and Turing. Several approaches of computation were proposed: recursive functions, Post programs, lambda calculus, Turing machines, register machines, to mention some. It was shown that all these models were equivalent in terms of its computability power. In this paper a new computational paradigm called arithmetic Petri nets (APN) is introduced. APN are Petri nets with inhibitor arcs that perform three types of arithmetical operations; increment, decrement and test for zero. Its equivalence to Turing and therefore to all the other approaches was shown to be true by simulation of a register machine. In this paper a different path is proposed by first proving that all recursive functions are APN computable. The opposite implication is obtained from Kleene's normal form theorem for APN. The normal form theorem was introduced by Kleene for the general recursive computation paradigm in terms of system of equations. The proof for the APN computational paradigm is based on the idea of computation tree, where each node of such a tree will tell us how a value needed for the arithmetical operations can be inductively obtained. Then, using arithmetization plus course of value recursion a primitive recursive predicate is defined from which by means of a primitive recursive function the desired characterization for the value of the output function is established.

AMS (MOS) Subject Classification: 03D78, 03D60, 03D20, 03D10.

Key words: Computability Power, Petri Nets, Inhibitor Arcs, Arithmetical Petri Nets, Normal Form Theorem, Arithmetization, Recursive Functions, Computation Tree.

1. Introduction

The notion of computability has its origin in the works of the logicians Church, Gödel, Kleene, Post and Turing. Several approaches of computation were proposed: recursive functions, Post programs, lambda calculus, Turing machines, register machines, to mention some. It was shown that all these models were equivalent in terms of its computability power. In this paper a new computational paradigm called arithmetic Petri nets (APN) is introduced. APN are Petri nets with inhibitor arcs (an extension of Petri nets, see [3], and [7], [8] for some modeling applications) that

Received December 12, 2015

perform three types of arithmetical operations; increment, decrement and test for zero.

It is a well known result that APN are equivalent to Turing machines and therefore inherit all their properties. The equivalence proof follows by noticing that APN simulate register machines and register machines, result to be Turing equivalent, as was shown by Sheperdson and Sturgis (see [2]). However, this is not for free, an increase in computational power, results in a decrease in decision power. In this paper a different path is proposed by first proving that all recursive functions are APN computable. The opposite implication is obtained from Kleene's normal form theorem for APN. The normal form theorem was introduced by Kleene for the general recursive computation paradigm in terms of system of equations [1]. The proof for the APN computational paradigm is based on the idea of computation tree, where each node of such a tree will tell us how a value needed for the arithmetical operations can be inductively obtained. Then, using arithmetization plus course of value recursion a primitive recursive predicate is defined from which by means of a primitive recursive function the desired characterization for the value of the output function is established. The proposed approach results to be original and elegant. In general, the equivalence between APN and Turing machines is given for granted, however is not easy to find a proof of it, even today with the power of the Internet. The paper is organized as follows: in section 2, all the preliminaries and information needed to understand and access the proof are given. It is also shown that all recursive functions are APN computable. In section three, the opposite implication is shown to be true. Some consequences are also addressed.

2. **Preliminaries**

NOTATION: $N = \{0, 1, 2, \dots\}$, $N^+_{n_0} = \{n_0, n_0 + 1, \dots, n_0 + k, \dots\}, n_0 \geq 0$.

A Petri net (PN) is a 5-tuple, $PN = \{P, T, F, W, M_0\}$ where: $P = \{p_1, p_2, \dots, p_m\}$ is a finite set of places, $T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions (represented respectively by circles and bars), $F \subset (P \times T) \cup (T \times P)$ is a set of arcs, $W : F \to N^+_1$ is a weight function, M_0: $P \to N$ is the initial marking, $P \cap T = \varnothing$ and $P \cup T \neq \varnothing$. Notice that if $W(p, t) = \alpha$ or $W(t, p) = \beta$, then, this is often represented graphically by α or β arcs from p to t or t to p (through arrowheads) each with no numeric label. Let $M_k(p_i)$ denote the marking (i.e., the number of tokens) at place $p_i \in P$ at time k and let $M_k = [M_k(p_1), \dots, M_k(p_m)]^T$ denote the marking (state) of PN at time k. A transition $t_j \in T$ is said to be enabled at time k if $M_k(p_i) \geq W(p_i, t_j)$ for all $p_i \in P$ such that $(p_{i,}t_j) \in F$. If a transition is enabled then, it can fire. If an enabled transition $t_j \in T$ fires at time k then, the next marking for $p_i \in P$ is given by

$$M_{k+1}(p_i) = M_k(p_i) + W(t_j, p_i) - W(p_i, t_j). \tag{2.1}$$

Let $A = [a_{ij}]$ denote an $n \times m$ matrix of integers (the incidence matrix) where $a_{ij} = a_{ij}^{+} - a_{ij}^{-}$ with $a_{ij}^{+} = W(t_i, p_j)$ and $a_{ij}^{-} = W(p_j, t_i)$. Let $u_k \in \{0,1\}^n$ denote a firing vector where if $t_j \in T$ is fired then, its corresponding firing vector is $u_k = [0, \ldots, 0, 1, 0, \ldots, 0]^T$ with the one in the j^{th} position in the vector and zeros everywhere else. The non-linear difference matrix equation describing the dynamical behavior represented by a PN is:

$$M_{k+1} = M_k + A^T u_k \tag{2.2}$$

where if at step k, $a_{ij}^{-} < M_k(p_j)$ for all $p_i \in P$ then, $t_i \in T$ is enabled and if this $t_i \in T$ fires then, its corresponding firing vector u_k is utilized in the difference equation to generate the next step. Notice that if M' can be reached from some other marking M and, if we fire some sequence of d transitions with corresponding firing vectors $u_0, u_1, \ldots, u_{d-1}$ we obtain that

$$M' = M + A^T u, \quad u = \sum_{k=0}^{d-1} u_k. \tag{2.3}$$

A Petri net with inhibitor arcs (PNI) is an extension of a PN where now there exists the possibility of connecting a place with a transition through an arc, where the arrowhead has been substituted by a small circle. A new firing rule is allowed, a transition is enabled if the marking of the places corresponding to inhibitor arcs are empty and as a consequence a new marking M_{k+1} will be generated.

Definition 1. An arithmetic Petri net (APN) consists of a PNI that executes the following three arithmetical operations: increment by one, decrement by one and testing for zero, as is depicted in Fig. 1.

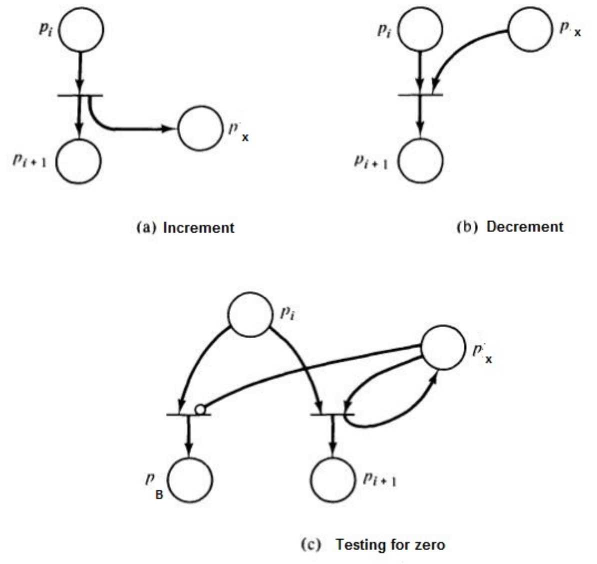


Figure 1. APN that executes increment. decrement and test for zero

Remark 2. Notice that the place p_i is the one that controls the execution of the arithmetical operation.

Definition 3. A partial function f on N is simply a function whose domain is a subset of N. If f is a partial function on N and $a \epsilon N$, then we write $f(a) \downarrow$, and say that $f(a)$ is defined to indicate that a is in the domain of f; if a is not in the domain of f, we write $f(a) \uparrow$ and say that $f(a)$ is undefined. If a partial function on N has the domain N, then it is called total.

Definition 4. A function $f(x_1, x_2, \ldots, x_m)$ (from $N^m \to N$) is APN computable if there exists an APN with an output place p_y, a set of input places $P_I = \{p_{x_1}, p_{x_2}, \ldots, p_{x_m}\}$, a set of auxiliary places P_A (as many as needed), not necessarily disjoint such that when the initial marking of the input places is set equal to the values of the variables of the function f, and all the other markings are assumed to be set to zero except for the places that control the execution of the arithmetical operations. A final marking is reached with the marking of p_y equal to the value of f i.e., $M_k(p_y) = f(x_1, x_2, \ldots, x_m)$. In the case when the APN does not reach a final marking i.e., there exists an infinite sequence of markings $M_1, M_2, \ldots$ beginning from the initial marking, then $f(x_1, x_2, \ldots, x_m)$ will be undefined denoted as $f(x_1, x_2, \ldots, x_m) \uparrow$.

Remark 5. It is supposed that there are always enough tokens in the places that control the execution of the arithmetical operations. This is similar to the execution of a software program where you go from one instruction to another or the same one. Therefore an APN computation behaves as a software program (see [5]). This similarity is of great help in understanding the logic in what is next presented.

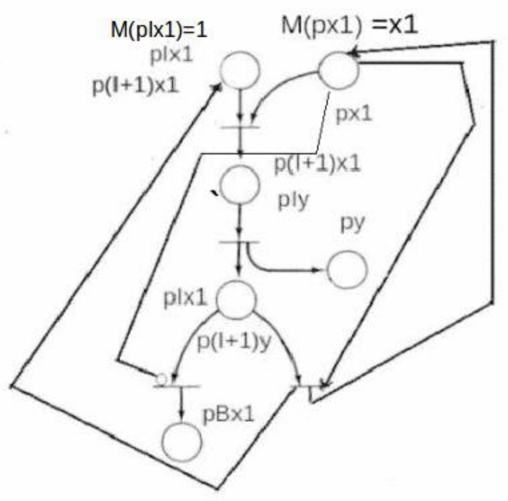


Figure 2. APN that computes the identity function

Example 6. The identity function is APN computable as is depicted in Fig. 2, and which is constituted by three blocks each one representing the decrement, increment and test for zero arithmetic operations of the APN.

Remark 7. Notice that a place can play two (or more) roles in a computation depending on the arithmetical operation that is being executed, as occurs in this example. It is also important to point out that we do not need such a complicated APN in case our only goal were to replicate at the output the input, since this can be easily achieved with a PN formed by two places and one transition.

Definition 8. A partial function f on N is simply a function whose domain is a subset of N. If f is a partial function on N and $a\epsilon N$, then we write $f(a)\downarrow$, and say that $f(a)$ is defined to indicate that a is in the domain of f; if a is not in the domain of f, we write $f(a)\uparrow$ and say that $f(a)$ is undefined. If a partial function on N has the domain N, then it is called total.

The following list of functions are called initial functions:

- The successor function $s(x) = x + 1$
- The nullity function $n(x) = 0$
- The projection function $u_i(x_1, x_2, \ldots, x_n) = x_i$

Definition 9. Let f be a function of k variables and let $g_1, \ldots, g_k$ be functions of n variables. Let

$$h(x_1, x_2, \ldots, x_n) = f(g_1(x_1, x_2, \ldots, x_n), g_2(x_1, x_2, \ldots, x_n), \ldots, g_k(x_1, x_2, \ldots, x_n)).$$

Then h is said to be obtained from f and $g_1, \ldots, g_k$ by composition.

Definition 10. The function h of $n+1$ variables is said to be obtained by primitive recursion, or simply recursion, from the total functions f (of n variables) and g (of $n+2$ variables) if the following iterative scheme holds,

$$h(x_1, x_2, \ldots, x_n, 0) = f(x_1, x_2, \ldots, x_n)$$

$$h(x_1, x_2, \ldots, x_n, t+1) = g(t, h(x_1, x_2, \ldots, x_n, t), x_1, x_2, \ldots, x_n).$$

Definition 11. The function f is said to be defined from the function g by minimalization if $\forall(x_1, x_2, \ldots, x_n)\exists y(g(x_1, x_2, \ldots, x_n, y) = 0)$ and $f(x) = \mu_y(g(x_1, x_2, \ldots, x_n, y) = 0)$, where $\mu_y(g(x_1, x_2, \ldots, x_n, y) = 0)$ is the least number y such that $g(x_1, x_2, \ldots, x_n, y) = 0$ holds. The function f is said to be defined from the total predicate P by minimalization if $f(x) = \mu_y(P(x_1, x_2, \ldots, x_n, y))$, where $\mu_y(P(x_1, x_2, \ldots, x_n, y))$ is the least number y such that $P(x_1, x_2, \ldots, x_n, y)$ holds.

Definition 12. A function f is primitive recursive if it can be obtained by a finite number of applications of composition and recursion, beginning with the initial functions.

Definition 13. A function f is partial recursive if it can be obtained by a finite number of applications of composition, recursion and minimalization, beginning with the initial functions.

Definition 14. A function f is recursive if it can be obtained by a finite number of applications of composition, recursion and minimalization of total functions, beginning with the initial functions.

In particular every primitive recursive function is recursive (for a proof see [5] and/or [6]). Notice that if f is recursive then it is partial recursive and total.

Definition 15. A predicate is said to be primitive primitive recursive (partial recursive, recursive) if its characteristic function is primitive recursive (partial recursive, recursive).

Next, it is proven that every recursive function is APN computable.

Theorem 16. *Let f be a recursive function then f is APN computable.*

Proof. This proof follows exactly the same lines given for flowcharts discussed in [4], therefore just an outline is next presented. By induction on the definition of recursive functions, first, it is proven to be true that the initial functions are APN computable. That the successor is, it is immediate, using the increment by one function. That the nullity is, follows using the decrement by one function, which will be operating until all the tokens in p_x are used and the marking of p_x is zero (see Fig. 1b). Finally, for the projection function use the APN given in Example 6, i.e., the identity function, for the x_i variable. Now lets suppose that for each case i.e., composition, recursion and minimalization, a set of functions which are APN computable have been obtained. Then, based on the diagrams presented in pages (66) and (67) of [4] (representing an APN computation as a block) the resulting function derived by applying composition, recursion and minimalization is APN computable. □

3. The Normal Form Theorem for APN

Arithmetization is a method of translating reasoning in natural language by arithmetical propositions, with the goal of substituting arguments with computations. Primitive recursive predicates and functions are the paradigm used to carry out arithmetizations [4]. Prime numbers $\Pi_{i\epsilon I}$ and factorizations are used for coding since they are well known to be primitive recursive. The code assigned to the sequence $[x_1, x_2, \ldots, x_n]$ is defined as:

$$[x_1, x_2, \ldots, x_n] = \Pi_0^n \cdot \Pi_1^{x_1} \cdots \Pi_n^{x_n},$$

and the decoding is given by: $[x]_n = \exp(x, \Pi_n)$, $ln(x) = (x)_0$ and $Seq(x) \Leftrightarrow (\forall n \leq x)(n > 0 \wedge (x)_n \neq 0 \rightarrow n \leq ln(x))$, where $ln(x)$ is the length of x and $(x)_n$ is the n-th component of x. Th concatenation of two sequences $*$ is defined to be $[x_0, x_1, \dots, x_n] * [y_1, y_2, \dots, y_m] = [x_1, x_2, \dots, x_n, y_1, y_2, \dots, y_m]$. Next, the following result which is basic for the proof of the normal form theorem, is given without proof which can be find in [4].

Proposition 17 (Course of values recursion)**.** *The class of primitive recursive functions is closed under recursions in which the definition of $f(x_1, x_2, \dots, x_n, y+1)$ may involve not just the last value $f(x_1, x_2, \dots, x_n, y)$ but any number of (and possibly all) values $f(x_1, x_2, \dots, x_n, z)_{z \leq y}$ already obtained. Formally, let $\hat{f}$ be the history function of f defined as:*

$$\hat{f}(x_1, x_2, \dots, x_n, y) = [f(x_1, x_2, \dots, x_n, 0), \dots, f(x_1, x_2, \dots, x_n, y)].$$

Then if f is defined as:

$$f(x_1, x_2, \dots, x_n, 0) = g(x_1, x_2, \dots, x_n),$$

$$f(x_1, x_2, \dots, x_n, y+1) = h(x_1, x_2, \dots, x_n, y,$$

$\hat{f}(x_1, x_2, \dots, x_n, y))$ and g, h are primitive recursive, so is f.

Theorem 18 (Normal Form Theorem for APN)**.** *Let $f(x_1, x_2, \dots, x_n)$ be APN computable, then there is a primitive recursive function U, a primitive recursive predicate T_n and a number e such that,*

$$f(x_1, x_2, \dots, x_n) = U(\mu_w T_n(e, x_1, x_2, \dots, x_n, w)).$$

Proof. The idea of the proof based on the notion of computation tree has been borrowed from the beautiful book of Odifreddi [4]. A computation tree allows us to organize the values of a given APN computable function f in a natural way. Each node of such a tree will tell us how a value needed in the computation can be inductively obtained by means of the APN arithmetical operations. The proof will be presented in a number of steps.

- Step 1. Definition of the APN.
  The APN will consist of: (1) a set of input places, denoted by $\{p_{x_i}\}_{i \epsilon N}$ whose elements correspond to each one of the input variables $x_1, x_2, \dots, x_n$ with $n \epsilon N$, an output place denoted by p_y, a set of auxiliary places, denoted by $\{p_{z_i}\}_{i=1}^{m}$ which correspond to variables possibly needed to compute f, a set of places associated to each one of the arithmetical operations performed over the input places and/or the auxiliary places and/or the output place (see the figures given in definition 1) denoted by $\{p_{I_{x_i \vee z_i \vee y}}\}_{i \epsilon N}$, $\{p_{(I+1)_{x_i \vee z_i \vee y}}\}_{i \epsilon N}$, and $\{p_{B_{x_i \vee z_i \vee y}}\}_{i \epsilon N}$, and (2) a set of transitions. (This notation was used in Example 6.)
- Step 2. Associate numbers to the places and to the arithmetical operations of the APN.
  To each one of the places, a unique numeral is given which will be denoted by

$\sharp p$, where p belongs to anyone of the set of places defined in step 1. In case of belonging to more than one, a unique number will be assigned which will be taken as the number that represents the place for the other cases. A numeral will also be assigned to each one of the arithmetical operations performed by the APN which will be denoted by $\sharp A$ and in particular, $\sharp Inc$, $\sharp Dec$ and $\sharp B$, depending on each case.

- Step 3. Associate numbers to the computations.
  This is done by induction on the construction of the computation tree. First of all, we assign numbers to nodes:

$$\sharp v = [[\sharp A, \sharp p_{I_{x_i \vee z_i \vee y}}, M_k(p_{I_{x_i \vee z_i \vee y}}), \sharp p_{(I+1)_{x_i \vee z_i \vee y}},$$
$$M_k(p_{(I+1)_{x_i \vee z_i \vee y}}), \sharp p_{B_{x_i \vee z_i \vee y}}, M_k(p_{B_{x_i \vee z_i \vee y}})],$$
$$[M_k(p_{x_1}), \dots, M_k(p_{x_n})], [M_k(p_{z_1}), \dots, M_k(p_{z_m})], M_k(p_y)]$$

  Thus a node is represented by four numbers: corresponding respectively to

$$e = [\sharp A, \sharp p_{I_{x_i \vee z_i \vee y}}, M_k(p_{I_{x_i \vee z_i \vee y}}), \sharp p_{(I+1)_{x_i \vee z_i \vee y}},$$
$$M_k(p_{(I+1)_{x_i \vee z_i \vee y}}), \sharp p_{B_{x_i \vee z_i \vee y}}, M_k(p_{B_{x_i \vee z_i \vee y}})],$$

  called the index at time k, the inputs, the auxiliary variables and the output.

Remark 19. A node gives us an instantaneous complete description of the state and the type of arithmetical operation of the APN at time k. Uniqueness in the number assigned to the node due to the fundamental theorem of arithmetic plus uniqueness in the number assigned to the places, is of great importance for decoding purposes i.e., given a fixed number, the topological representation of the node which consists of the type of instruction and the places that are involved as well as its markings can be recovered, whenever the representation obtained from the number makes sense in terms of the definition of the arithmetical operations.

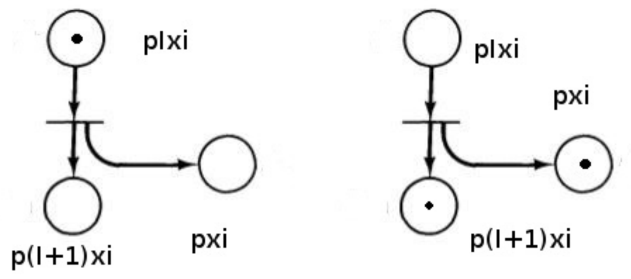
We then assign numbers to trees: each tree T consists of a node v with associated number $\sharp v$, and of a certain number (finite, and possibly equal to zero) of ordered predecessors, each one being a sub-tree T_i. By induction, we assign to this tree the number

$$\sharp T = [\sharp v, \sharp T_1, \dots, \sharp T_m],$$

where $\sharp T_i$ is the number assigned to the sub-tree T_i i.e., $\sharp T_i = i + 1$. In particular, if the vertex with number $\sharp v$ does not have predecessors, then it has number $[\sharp v]$ as a tree.

- Step 4. Translate in a primitive recursive predicate $T(w)$ the property that w is a number coding a computation tree.

Remark 20. To increase readability we use commas instead of nested parentheses, and write e.g. $(a)_{i,j,k}$ in place of $(((a)_i)_j)_k)$.



(A) Initial State (B) Final State

Fig 3. Increment

Setting $w = [\sharp v, \sharp T_1, \ldots, \sharp T_m]$ where:

1. $(w)_1 = [e, [M_k(p_{x_1}), \ldots, M_k(p_{x_n})], [M_k(p_{z_1}), \ldots, M_k(p_{z_m})], M_k(p_y)]$
2. $(w)_{1,1} = e$
3. $(w)_{1,2} = [M_k(p_{x_1}), \ldots, M_k(p_{x_n})]$
4. $(w)_{1,3} = [M_k(p_{z_1}), \ldots, M_k(p_{z_m})]$
5. $(w)_{1,4} = M_k(p_y)$
6. $(w)_{i+1} = \sharp T_i$
7. $(w)_{i+1,1} =$ number associated to the node corresponding to T_i

(which are a direct consequence of the definitions given in step 3). First, it is assured that w is well defined i.e.,
$Const(w) \Leftrightarrow Seq(w) \wedge Seq((w)_1) \wedge ln((w)_1) = 4 \wedge Seq((w)_{1,1}) \wedge ln((w)_{1,1}) = 7 \wedge Seq((w)_{1,2}) \wedge ln((w)_{1,2}) = n, \wedge Seq((w)_{1,3}) \wedge ln((w)_{1,3}) = m$

Now, let's take care of the three possible arithmetic operations performed by the APN.

Increment: From the description given in Fig. 3, there are three cases to be considered:

Increment in the input:

$$(w)_{2,1} = [[\sharp Inc, \sharp p_{I_{x_i}}, M_k(p_{I_{x_i}}), \sharp p_{(I+1)_{x_i}}, M_k(p_{(I+1)_{x_i}}), 0, 0],$$
$$[M_k(p_{x_1}), \ldots, M_k(p_{x_i}) \ldots, M_k(p_{x_n})],$$

$$[M_k(p_{z_1}), \ldots, M_k(p_{z_m})], M_k(p_y)] \rightarrow (w)_1 = [[\sharp Inc, \sharp p_{I_{x_i}}, M_{k+1}(p_{I_{x_i}}) = M_k(p_{I_{x_i}}) - 1,$$
$$\sharp p_{(I+1)_{x_i}}, M_{k+1}(p_{(I+1)_{x_i}}) = M_k(p_{(I+1)_{x_i}}) + 1, 0, 0],$$

$$[M_{k+1}(p_{x_1}), \ldots, M_{k+1}(p_{x_i})$$
$$= M_k(p_{x_i}) + 1, \ldots, M_{k+1}(p_{x_n})], [M_{k+1}(p_{z_1}), \ldots, M_k(p_{z_m})], M_{k+1}(p_y)]$$

Increment in the auxiliary variables:

$$(w)_{2,1} = [[\sharp Inc, \sharp p_{I_{x_i}}, M_k(p_{I_{x_i}}), \sharp p_{(I+1)_{x_i}}, M_k(p_{(I+1)_{x_i}}), 0, 0], [M_k(p_{x_1}), \ldots, M_k(p_{x_n})],$$

$$[M_k(p_{z_1}),\ldots,M_k(p_{z_i}),\ldots,M_k(p_{z_m})],M_k(p_y)]\rightarrow(w)_1$$
$$=[[\sharp Inc,\sharp p_{I_{x_i}},M_{k+1}(p_{I_{x_i}})=M_k(p_{I_{x_i}})-1,$$
$$\sharp p_{(I+1)_{x_i}},M_{k+1}(p_{(I+1)_{x_i}})=M_k(p_{(I+1)_{x_i}})+1,0,0],$$
$$[M_{k+1}(p_{x_1}),\ldots,M_{k+1}(p_{x_n})],[M_{k+1}(p_{z_1}),\ldots,M_{k+1}(p_{z_i})$$
$$=M_k(p_{z_i})+1,\ldots,M_k(p_{z_m})],M_{k+1}(p_y)]$$

Increment in the output:

$$(w)_{2,1}=[[\sharp Inc,\sharp p_{I_{x_i}},M_k(p_{I_{x_i}}),\sharp p_{(I+1)_{x_i}},M_k(p_{(I+1)_{x_i}}),0,0],[M_k(p_{x_1}),\ldots,M_k(p_{x_n})],$$
$$[M_k(p_{z_1}),\ldots,M_k(p_{z_m})],M_k(p_y)]\rightarrow(w)_1=[[\sharp Inc,\sharp p_{I_{x_i}},M_{k+1}(p_{I_{x_i}})=M_k(p_{I_{x_i}})-1,$$
$$\sharp p_{(I+1)_{x_i}},M_{k+1}(p_{(I+1)_{x_i}})=M_k(p_{(I+1)_{x_i}})+1,0,0],$$
$$[M_{k+1}(p_{x_1}),\ldots,M_{k+1}(p_{x_n})],[M_{k+1}(p_{z_1}),\ldots,M_k(p_{z_m})],M_{k+1}(p_y)=M_k(p_y)+1]$$

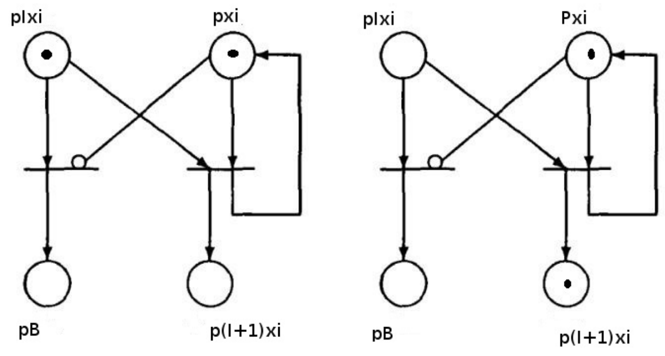
Therefore we get that:
$Inc(w)\Leftrightarrow[ln(w)=2\wedge ln(w)_{1,1}=ln(w)_{2,1,1}\wedge(w)_{1,1,1}=(w)_{2,1,1,1}\wedge(w)_{1,1,2}=(w)_{2,1,1,2}\wedge(w)_{1,1,3}=(w)_{2,1,1,3}-1\wedge(w)_{1,1,4}=(w)_{2,1,1,4}\wedge(w)_{1,1,5}=(w)_{2,1,1,5}+1\wedge(w)_{1,1,6}=(w)_{2,1,1,6}\wedge(w)_{1,1,7}=(w)_{2,1,1,7}\wedge ln(w)_{1,2}=ln(w)_{2,1,2}\wedge(w)_{1,2,1}=(w)_{2,1,2,1}\wedge\ldots\wedge(w)_{1,2,i}=(w)_{2,1,2,i}+1\wedge\ldots\wedge(w)_{1,2,n}=(w)_{2,1,2,n}\wedge ln(w)_{1,3}=ln(w)_{2,1,3}\wedge(\forall)_{i=1}^{m}((w)_{1,3,i}=(w)_{2,1,3,i})\wedge(w)_{1,4}=(w)_{2,1,4}]\vee[ln(w)=2\wedge ln(w)_{1,1}=ln(w)_{2,1,1}\wedge(w)_{1,1,1}=(w)_{2,1,1,1}\wedge(w)_{1,2,2,2}=(w)_{2,1,2}\wedge(w)_{1,1,3}=(w)_{2,1,1,3}-1\wedge(w)_{1,1,4}=(w)_{2,1,1,4}\wedge(w)_{1,1,5}=(w)_{2,1,1,5}+1\wedge(w)_{1,1,6}=(w)_{2,1,1,6}\wedge(w)_{1,1,7}=(w)_{2,1,1,7}\wedge ln(w)_{1,2}=ln(w)_{2,1,2}\wedge(\forall)_{i=1}^{n}((w)_{1,2,i}=(w)_{2,1,2,i})\wedge ln(w)_{1,3}=ln(w)_{2,1,3}\wedge(w)_{1,3,1}=(w)_{2,1,2,1}\wedge\ldots\wedge(w)_{1,3,i}=(w)_{2,1,3,i}+1\wedge\ldots\wedge(w)_{1,3,m}=(w)_{2,1,3,m}\wedge(w)_{1,4}=(w)_{2,1,4}]\vee[ln(w)=2\wedge ln(w)_{1,1}=ln(w)_{2,1,1}\wedge(w)_{1,1,1}=(w)_{2,1,1,1}\wedge(w)_{1,2,2,2}=(w)_{2,1,2}\wedge(w)_{1,1,3}=(w)_{2,1,1,3}-1\wedge(w)_{1,1,4}=(w)_{2,1,1,4}\wedge(w)_{1,1,5}=(w)_{2,1,1,5}+1\wedge(w)_{1,1,6}=(w)_{2,1,1,6}\wedge(w)_{1,1,7}=(w)_{2,1,1,7}\wedge ln(w)_{1,2}=ln(w)_{2,1,2}\wedge(\forall)_{i=1}^{n}((w)_{1,2,i}=(w)_{2,1,,i})\wedge ln(w)_{1,3}=ln(w)_{2,1,3}\wedge(\forall)_{i=1}^{m}((w)_{1,3,i}=(w)_{2,1,3,i})\wedge(w)_{1,4}=(w)_{2,1,4}+1]$

Decrement: This case is exactly the same as the previous one just change plus one by minus one for each one of the input, auxiliary and output places. The description is denoted by $Dec(w)$.

Test for zero: there are two cases: no branching (Fig. 4) and branching (Fig. 5). Each one with three sub-cases to be considered:

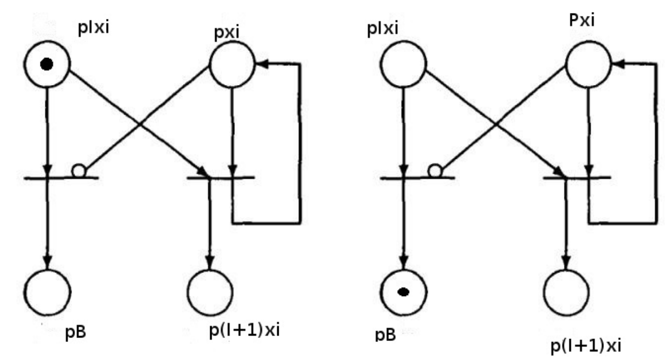
The condition is not satisfied (no branching):
Input:

$$(w)_{2,1}=[[\sharp B,\sharp p_{I_{x_i}},M_k(p_{I_{x_i}}),\sharp p_{(I+1)_{x_i}},M_k(p_{(I+1)_{x_i}}),p_{B_{x_i}},M_k(p_{B_{x_i}})],$$
$$[M_k(p_{x_1}),\ldots,M_k(p_{x_i})>0,\ldots,M_k(p_{x_n})],[M_k(p_{z_1}),\ldots,M_k(p_{z_m})],M_k(p_y)]\rightarrow$$
$$(w)_1=[[\sharp B,\sharp p_{I_{x_i}},M_{k+1}(p_{I_{x_i}})=M_k(p_{I_{x_i}})-1,$$
$$\sharp p_{(I+1)_{x_i}},M_{k+1}(p_{(I+1)_{x_i}})=M_k(p_{(I+1)_{x_i}})+1,\sharp p_{B_{x_i}},M_{k+1}(p_{B_{x_i}})=M_k(p_{B_{x_i}})],$$



(C) Initial State (D) Final State

Fig 4. No branching



(E) Initial State (F) Final State

Fig 5. Branching

$$[M_{k+1}(p_{x_1}), \ldots, M_{k+1}(p_{x_i})$$
$$= M_k(p_{x_i}), \ldots, M_{k+1}(p_{x_n})], [M_{k+1}(p_{z_1}), \ldots, M_k(p_{z_m})], M_{k+1}(p_y)]$$

Auxiliary variables

$$(w)_{2,1} = [[\sharp B, \sharp p_{I_{x_i}}, M_k(p_{I_{x_i}}), \sharp p_{(I+1)_{x_i}}, M_k(p_{(I+1)_{x_i}}), p_{B_{x_i}}, M_k(p_{B_{x_i}})],$$
$$[M_k(p_{x_1}), \ldots, M_k(p_{x_n})],$$
$$[M_k(p_{z_1}), \ldots, M_k(p_{z_i}) > 0, \ldots, M_k(p_{z_m})], M_k(p_y)] \rightarrow$$
$$(w)_1 = [[\sharp B, \sharp p_{I_{x_i}}, M_{k+1}(p_{I_{x_i}}) = M_k(p_{I_{x_i}}) - 1,$$
$$\sharp p_{(I+1)_{x_i}}, M_{k+1}(p_{(I+1)_{x_i}}) = M_k(p_{(I+1)_{x_i}}) + 1, \sharp p_{B_{x_i}}, M_{k+1}(p_{B_{x_i}}) = M_k(p_{B_{x_i}})],$$
$$[M_{k+1}(p_{x_1}), \ldots, M_{k+1}(p_{x_n})], [M_{k+1}(p_{z_1}), \ldots, M_{k+1}(p_{z_i}) = M_k(p_{z_i}), \ldots, M_k(p_{z_m})], M_{k+1}(p_y)]$$

Output

$$(w)_{2,1} = [[\sharp B, \sharp p_{I_{x_i}}, M_k(p_{I_{x_i}}), \sharp p_{(I+1)_{x_i}},$$
$$M_k(p_{(I+1)_{x_i}}), p_{B_{x_i}}, M_k(p_{B_{x_i}})], [M_k(p_{x_1}), \ldots, M_k(p_{x_n})],$$
$$[M_k(p_{z_1}), \ldots, M_k(p_{z_m})], M_k(p_y) > 0] \rightarrow (w)_1 = [[\sharp B, \sharp p_{I_{x_i}}, M_{k+1}(p_{I_{x_i}}) = M_k(p_{I_{x_i}}) - 1,$$

$$\sharp p_{(I+1)_{x_i}}, M_{k+1}(p_{(I+1)_{x_i}}) = M_k(p_{(I+1)_{x_i}}) + 1, \sharp p_{B_{x_i}}, M_{k+1}(p_{B_{x_i}}) = M_k(p_{B_{x_i}})],$$
$$[M_{k+1}(p_{x_1}), \ldots, \ldots, M_{k+1}(p_{x_n})], [M_{k+1}(p_{z_1}), \ldots, M_k(p_{z_m})], M_{k+1}(p_y) = M_k(p_y)]$$

The condition is satisfied (branching):

Input:

$$(w)_{2,1} = [[\sharp B, \sharp p_{I_{x_i}}, M_k(p_{I_{x_i}}), \sharp p_{(I+1)_{x_i}}, M_k(p_{(I+1)_{x_i}}), p_{B_{x_i}}, M_k(p_{B_{x_i}})],$$
$$[M_k(p_{x_1}), \ldots, M_k(p_{x_i}) = 0, \ldots, M_k(p_{x_n})], [M_k(p_{z_1}), \ldots, M_k(p_{z_m})], M_k(p_y)] \rightarrow$$
$$(w)_1 = [[\sharp B, \sharp p_{I_{x_i}}, M_{k+1}(p_{I_{x_i}}) = M_k(p_{I_{x_i}}) - 1,$$
$$\sharp p_{(I+1)_{x_i}}, M_{k+1}(p_{(I+1)_{x_i}}) = M_k(p_{(I+1)_{x_i}}), \sharp p_{B_{x_i}}, M_{k+1}(p_{B_{x_i}}) = M_k(p_{B_{x_i}}) + 1],$$
$$[M_{k+1}(p_{x_1}), \ldots, M_{k+1}(p_{x_i}) = M_k(p_{x_i}), \ldots, M_{k+1}(p_{x_n})],$$
$$[M_{k+1}(p_{z_1}), \ldots, M_k(p_{z_m})], M_{k+1}(p_y)]$$

Auxiliary variables

$$(w)_{2,1} = [[\sharp B, \sharp p_{I_{x_i}}, M_k(p_{I_{x_i}}), \sharp p_{(I+1)_{x_i}}, M_k(p_{(I+1)_{x_i}}), p_{B_{x_i}}, M_k(p_{B_{x_i}})],$$
$$[M_k(p_{x_1}), \ldots, M_k(p_{x_n})],$$
$$[M_k(p_{z_1}), \ldots, M_k(p_{z_i}) = 0, \ldots, M_k(p_{z_m})], M_k(p_y)] \rightarrow$$
$$(w)_1 = [[\sharp B, \sharp p_{I_{x_i}}, M_{k+1}(p_{I_{x_i}}) = M_k(p_{I_{x_i}}) - 1,$$
$$\sharp p_{(I+1)_{x_i}}, M_{k+1}(p_{(I+1)_{x_i}}) = M_k(p_{(I+1)_{x_i}}), \sharp p_{B_{x_i}}, M_{k+1}(p_{B_{x_i}}) = M_k(p_{B_{x_i}}) + 1],$$
$$[M_{k+1}(p_{x_1}), \ldots, M_{k+1}(p_{x_n})], [M_{k+1}(p_{z_1}), \ldots, M_{k+1}(p_{z_i}) = M_k(p_{z_i}), \ldots, M_k(p_{z_m})], M_{k+1}(p_y)]$$

Output

$$(w)_{2,1} = [[\sharp B, \sharp p_{I_{x_i}}, M_k(p_{I_{x_i}}), \sharp p_{(I+1)_{x_i}}, M_k(p_{(I+1)_{x_i}}), p_{B_{x_i}}, M_k(p_{B_{x_i}})],$$
$$[M_k(p_{x_1}), \ldots, M_k(p_{x_n})],$$
$$[M_k(p_{z_1}), \ldots, M_k(p_{z_m})], M_k(p_y) = 0] \rightarrow (w)_1 = [[\sharp B, \sharp p_{I_{x_i}}, M_{k+1}(p_{I_{x_i}}) = M_k(p_{I_{x_i}}) - 1,$$
$$\sharp p_{(I+1)_{x_i}}, M_{k+1}(p_{(I+1)_{x_i}}) = M_k(p_{(I+1)_{x_i}}), \sharp p_{B_{x_i}}, M_{k+1}(p_{B_{x_i}}) = M_k(p_{B_{x_i}}) + 1],$$
$$[M_{k+1}(p_{x_1}), \ldots, \ldots, M_{k+1}(p_{x_n})], [M_{k+1}(p_{z_1}), \ldots, M_k(p_{z_m})], M_{k+1}(p_y) = M_k(p_y)]$$

Therefore we get that:

$Tz(w) \Leftrightarrow [[[ln(w) = 2 \wedge ln(w)_{1,1} = ln(w)_{2,1,1} \wedge (w)_{1,1,1} = (w)_{2,1,1,1} \wedge (w)_{1,1,2} = (w)_{2,1,1,2} \wedge (w)_{1,1,3} = (w)_{2,1,1,3} - 1 \wedge (w)_{1,1,4} = (w)_{2,1,1,4} \wedge (w)_{1,1,5} = (w)_{2,1,1,5} + 1 \wedge (w)_{1,1,6} = (w)_{2,1,1,6} \wedge (w)_{1,1,7} = (w)_{2,1,1,7} \wedge ln(w)_{1,2} = ln(w)_{2,1,2} \wedge (w)_{1,2,1} = (w)_{2,1,2,1} \wedge \ldots \wedge (w)_{1,2,i} = (w)_{2,1,2,i} > 0 \wedge \ldots \wedge (w)_{1,2,n} = (w)_{2,1,2,n} \wedge ln(w)_{1,3} = ln(w)_{2,1,3} \wedge (\forall)_{i=1}^{m}((w)_{1,3,i} = (w)_{2,1,3,i}) \wedge (w)_{1,4} = (w)_{2,1,4}] \vee [ln(w) = 2 \wedge ln(w)_{1,1} = ln(w)_{2,1,1} \wedge (w)_{1,1,1} = (w)_{2,1,1,1} \wedge (w)_{1,1,2} = (w)_{2,1,1,2} \wedge (w)_{1,1,3} = (w)_{2,1,1,3} - 1 \wedge (w)_{1,1,4} = (w)_{2,1,1,4} \wedge (w)_{1,1,5} = (w)_{2,1,1,5} + 1 \wedge (w)_{1,1,6} = (w)_{2,1,1,6} \wedge (w)_{1,1,7} = (w)_{2,1,1,7} \wedge ln(w)_{1,2} = ln(w)_{2,1,2} \wedge (\forall)_{i=1}^{n}((w)_{1,2,i} = (w)_{2,1,2,i}) \wedge ln(w)_{1,3} = ln(w)_{2,1,3} \wedge (w)_{1,3,1} = (w)_{2,1,2,1} \wedge \ldots \wedge (w)_{1,3,i} = (w)_{2,1,3,i} > 0 \wedge \ldots \wedge (w)_{1,3,m} = (w)_{2,1,3,m} \wedge (w)_{1,4} = (w)_{2,1,4}] \vee [ln(w) = 2 \wedge ln(w)_{1,1} = ln(w)_{2,1,1} \wedge (w)_{1,1,1} = (w)_{2,1,1,1} \wedge (w)_{1,1,2} = (w)_{2,1,1,2} \wedge (w)_{1,1,3} =$

$(w)_{2,1,1,3} - 1 \wedge (w)_{1,1,4} = (w)_{2,1,1,4} \wedge (w)_{1,1,5} = (w)_{2,1,1,5} + 1 \wedge (w)_{1,1,6} = (w)_{2,1,1,6} \wedge (w)_{1,1,7} = (w)_{2,1,1,7} \wedge ln(w)_{1,2} = ln(w)_{2,1,2} \wedge (\forall)_{i=1}^{n}((w)_{1,2,i} = (w)_{2,1,,i}) \wedge ln(w)_{1,3} = ln(w)_{2,1,3} \wedge (\forall)_{i=1}^{m}((w)_{1,3,i} = (w)_{2,1,3,i}) \wedge (w)_{1,4} = (w)_{2,1,4} > 0]] \vee [[ln(w) = 2 \wedge ln(w)_{1,1} = ln(w)_{2,1,1} \wedge (w)_{1,1,1} = (w)_{2,1,1,1} \wedge (w)_{1,1,2} = (w)_{2,1,1,2} \wedge (w)_{1,1,3} = (w)_{2,1,1,3} - 1 \wedge (w)_{1,1,4} = (w)_{2,1,1,4} \wedge (w)_{1,1,5} = (w)_{2,1,1,5} \wedge (w)_{1,1,6} = (w)_{2,1,1,6} \wedge (w)_{1,1,7} = (w)_{2,1,1,7} + 1 \wedge ln(w)_{1,2} = ln(w)_{2,1,2} \wedge (w)_{1,2,1} = (w)_{2,1,2,1} \wedge \ldots \wedge (w)_{1,2,i} = (w)_{2,1,2,i} = 0 \wedge \ldots \wedge (w)_{1,2,n} = (w)_{2,1,2,n} \wedge ln(w)_{1,3} = ln(w)_{2,1,3} \wedge (\forall)_{i=1}^{m}((w)_{1,3,i} = (w)_{2,1,3,i}) \wedge (w)_{1,4} = (w)_{2,1,4}] \vee [ln(w) = 2 \wedge ln(w)_{1,1} = ln(w)_{2,1,1} \wedge (w)_{1,1,1} = (w)_{2,1,1,1} \wedge (w)_{1,1,2} = (w)_{2,1,1,2} \wedge (w)_{1,1,3} = (w)_{2,1,1,3} - 1 \wedge (w)_{1,1,4} = (w)_{2,1,1,4} \wedge (w)_{1,1,5} = (w)_{2,1,1,5} \wedge (w)_{1,1,6} = (w)_{2,1,1,6} \wedge (w)_{1,1,7} = (w)_{2,1,1,7} + 1 \wedge ln(w)_{1,2} = ln(w)_{2,1,2} \wedge (\forall)_{i=1}^{n}((w)_{1,2,i} = (w)_{2,1,2,i}) \wedge ln(w)_{1,3} = ln(w)_{2,1,3} \wedge (w)_{1,3,1} = (w)_{2,1,2,1} \wedge \ldots \wedge (w)_{1,3,i} = (w)_{2,1,3,i} = 0 \wedge \ldots \wedge (w)_{1,3,m} = (w)_{2,1,3,m} \wedge (w)_{1,4} = (w)_{2,1,4}] \vee [ln(w) = 2 \wedge ln(w)_{1,1} = ln(w)_{2,1,1} \wedge (w)_{1,1,1} = (w)_{2,1,1,1} \wedge (w)_{1,1,2} = (w)_{2,1,1.2} \wedge (w)_{1,1,3} = (w)_{2,1,1,3} - 1 \wedge (w)_{1,1,4} = (w)_{2,1,1,4} \wedge (w)_{1,1,5} = (w)_{2,1,1,5} \wedge (w)_{1,1,6} = (w)_{2,1,1,6} \wedge (w)_{1,1,7} = (w)_{2,1,1,7} + 1 \wedge ln(w)_{1,2} = ln(w)_{2,1,2} \wedge (\forall)_{i=1}^{n}((w)_{1,2,i} = (w)_{2,1,,i}) \wedge ln(w)_{1,3} = ln(w)_{2,1,3} \wedge (\forall)_{i=1}^{m}((w)_{1,3,i} = (w)_{2,1,3,i}) \wedge (w)_{1,4} = (w)_{2,1,4} = 0]]]$
which can be reduced to:
$Tz(w) \Leftrightarrow [[ln(w) = 2 \wedge ln(w)_{1,1} = ln(w)_{2,1,1} \wedge (w)_{1,1,1} = (w)_{2,1,1,1} \wedge (w)_{1,1,2} = (w)_{2,1,1,2} \wedge (w)_{1,1,3} = (w)_{2,1,1,3} - 1 \wedge (w)_{1,1,4} = (w)_{2,1,1,4} \wedge (w)_{1,1,5} = (w)_{2,1,1,5} + 1 \wedge (w)_{1,1,6} = (w)_{2,1,1,6} \wedge (w)_{1,1,7} = (w)_{2,1,1,7} \wedge ln(w)_{1,2} = ln(w)_{2,1,2} \wedge (w)_{1,2,1} = (w)_{2,1,2,1} \wedge \ldots \wedge (w)_{1,2,i} = (w)_{2,1,2,i} > 0 \wedge \ldots \wedge (w)_{1,2,n} = (w)_{2,1,2,n} \wedge ln(w)_{1,3} = ln(w)_{2,1,3} \wedge (\forall)_{i=1}^{m}((w)_{1,3,i} = (w)_{2,1,3,i}) \wedge (w)_{1,4} = (w)_{2,1,4} > 0] \vee [ln(w) = 2 \wedge ln(w)_{1,1} = ln(w)_{2,1,1} \wedge (w)_{1,1,1} = (w)_{2,1,1,1} \wedge (w)_{1,1,2} = (w)_{2,1,1,2} \wedge (w)_{1,1,3} = (w)_{2,1,1,3} - 1 \wedge (w)_{1,1,4} = (w)_{2,1,1,4} \wedge (w)_{1,1,5} = (w)_{2,1,1,5} \wedge (w)_{1,1,6} = (w)_{2,1,1,6} \wedge (w)_{1,1,7} = (w)_{2,1,1,7} + 1 \wedge ln(w)_{1,2} = ln(w)_{2,1,2} \wedge (w)_{1,2,1} = (w)_{2,1,2,1} \wedge \ldots \wedge (w)_{1,2,i} = (w)_{2,1,2,i} = 0 \wedge \ldots \wedge (w)_{1,2,n} = (w)_{2,1,2,n} \wedge ln(w)_{1,3} = ln(w)_{2,1,3} \wedge (\forall)_{i=1}^{m}((w)_{1,3,i} = (w)_{2,1,3,i}) \wedge (w)_{1,4} = (w)_{2,1,4} = 0]].$

This reduction could have also been derived by analyzing the APN for the test for zero case, where the only places whose marking changes are $p_{I_{x_i}}$ and $p_{(I+1)_{X_{i+1}}}$ (for the no-branching case), and $p_{I_{x_i}}$ and $p_{B_{X_i}}$ (for the branching case) with all the remaining places being unchanged.

Therefore, we may define inductively:

$$T(w) \Leftrightarrow Const(w) \wedge [Inc(w) \vee Dec(w) \vee Tz(w)] \wedge [(\forall)_{i=2}^{ln(w)} T((w)_i)]$$

which is by course of values induction primitive recursive.

- Step 5. Define T_n and U.
  Let:

$$T_n(e, x_1, \ldots, x_n, w) \Leftrightarrow T(w) \wedge (w)_{1,1} = e \wedge (w)_{ln(w),2} = [M_0(p_{x_1}), \ldots, M_0(p_{x_n})]$$

  and

$$U(w) = (w)_{1,4},$$

which are primitive recursive. Where T_n is the predicate that translates: w is the number of a computation tree of the value of the arithmetic operation with associated number e, on inputs $x_1, x_2, \dots, x_n$.

- Step 6. End of the proof.
  Let $f(x_1, x_2, \dots, x_n)$ be APN computable with number e. Since f is total, it is guarantee that for every set of values of the input variables there is a computation tree employing the computation procedure given by e. Therefore searching for the smallest tree i.,e., the one with the smallest code number and extracting the value of the function by looking at its fourth node number we get that:

$$f(x_1, x_2, \dots, x_n) = U(\mu_w T_n(e, x_1, x_2, \dots, x_n, w)).$$

□

The following corollary is a direct consequence of Theorems 16 and 18.

Corollary 21. *$f(x_1, x_2, \dots, x_n)$ is APN computable if and only if f is recursive.*

Finally, the claim that APN are equivalent to Turing machines is established.

Corollary 22. *$f(x_1, x_2, \dots, x_n)$ is APN computable if and only if f is Turing computable.*

Proof. It follows from the fact that recursive is equivalent to Turing computability [6]. □

CONSEQUENCES

- The normal form theorem is also true for APN partial computable functions. It is just enough to observe that $\mu_w T_n(k, x_1, x_2, \dots, x_n, w)$ is defined if and only if there exists an APN-computation, that is, if and only if f is defined.
- An enumeration of all partial computable functions through number e is obtained. In this sense the APN model of computation is universal.
- The number e is not unique since it can be increased by adding superfluous arithmetical operations. This is formally proved by the Padding lemma (see [4]).

REFERENCES

[1] S. Kleene. Introduction to Metamathematics. Noth Holland Publishing Co. Amsterdam, Groningen (1952).

[2] J. C. Sheperdzon and H. E. Sturgis. Computability of Recursive Functions. Journal of the ACM, vol 10, no 2, 1963.

[3] J. L. Peterson. Petri Net Theory and The Modelling of Systems. Prentice Hal 1981.

[4] P. Odifreddi. Recursion Theory, The theory of functions and sets of natural numbers. Studies in logic and the foundations of mathematics, Elsevier 1999.

[5] M. Davis, R. Sigal and E. Weyuker. Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science. Academic Press, 1983.

[6] M. Davis. Computability and Unsolvability. McGraw-Hill 1958.

[7] Z. Retchkiman. Modeling and Stability Analysis of fault queuing systems. Neural Parallel and Scientific Computations, (20) 2012.

[8] Z. Retchkiman. The modeling and stability problem for a communication network system. Neural Parallel and Scientific Computations, (22) 2014.