A Simple Positive Flows Computation Algorithm for a Large Subclass of Colored Nets

S. Evangelista, C. Pajault, and J.F. Pradat-Peyre

CEDRIC - CNAM Paris
292, rue St Martin, 75003 Paris
{evangeli,christophe.pajault,peyre}@cnam.fr

Abstract. Positive flows provide very useful informations that can be used to perform efficient analysis of a model. Although algorithms computing (a generative family of) positive flows in ordinary Petri nets are well known, computing a generative family of positive flows in colored net remains an open problem. We propose in this paper a pragmatic approach that allows us to define an algorithm that computes a generative family of particular but useful positive flows in a large subclass of colored nets: the simple well-formed nets.

1 Introduction

One of the principal reasons to use Petri nets to model distributed algorithms is that one can combine structural techniques (that use only the structure of the net) with model-checking techniques (that perform exhaustive simulations) in order to analyze a net and demonstrate properties. Among structural techniques, invariants computation can be viewed as the most fundamental one : invariants give immediate indication on the behavior of the model (i.e. without needing to "execute" it) ; invariants are needed to perform structural reductions (like the implicit place reduction [1], [14], or efficient transitions agglomeration [10, 17]); places or transitions invariants can be used to define/classify kind of Petri nets with simplified liveness conditions (e.g. flexible manufacturing systems) [13], and so on. Among invariants, it is well known that positive flows (flows that use only positive weights) are the most useful ones and give accurate information on the net. In ordinary Petri nets, invariants are computed with the Gauss algorithm when no positive constraint is added or with the Farkas algorithm when a generative family of positive flows is needed (both algorithms are described in [4]).

However, in many cases one uses colored Petri nets for modeling algorithms. Indeed, they allow a more concise description of a problem than ordinary Petri nets. Furthermore, defining parameterized models permits to study a set of solutions with a unique model. So, computing invariants in a colored net is an interesting challenge. However, these calculus raise new problems : we have to manipulate color mapping instead of integer values, models may be parameterized and then algorithms must tackle with this additional difficulties. When dealing with flows computation, two main approaches are used : generalizing Gauss algorithm to take into account color mapping (in particular with the use of the notion of "generalized inverse") [15], [20], [25], [7]; this first approach permits to obtain a generative family (for the last citation) but requires to fix parameters and computed flows are not easily usable. The second approach consists in restricting color nets; this lead to different algorithms that compute parameterized and useful flows in regular nets [16] or ordered and associative nets [8].

Nevertheless, only one algorithm is known today in the case of colored positive flows computation [5], and this algorithm works only on very restricted models : unary regular nets or unary predicate/transitions nets.

We propose in this paper an algorithm that computes particular, but useful, positive flows in a high level model : the simple well-formed nets. This model is a restriction of the general well-formed nets [3], but sufficient enough to model Ada programs [2]. This paper is organized as follow : after some definitions we show, in section 3, how we can take advantage of both syntactic restrictions used to define nets and positive flows to obtain a particular "fractal" and "regular" equations system. Then we propose an algorithm that computes a generative family of *simple* positive flows (note that this family does not generate *all* positive flows). At last we conclude and propose future possible extensions of this algorithm.

2 Definitions

2.1 Colored Petri nets

Petri nets form a well known formalism used to express and to analyze concurrent behaviours [21, 23]. However, it is often difficult to model complex problems because of the "low level" expression power provided by Petri nets. In practice, one uses colored nets, that are an abbreviation of Petri nets. This abbreviation is based on the idea to associate to the classical Petri net token a type (also called a color) that gives it a high level semantic : instead saying "there are three tokens in place p" one can say for instance "there are the token 1, the token 4 and the token 13 in place p". In this way, one can model complex synchronization schemes involving data carried by processes.

Definition 1 (Multi-sets). *A multi-set over a finite and non empty set C is an application from C to $\mathbb{N}$. We denote by $Bag_{\mathbb{N}}(C)$ (or $Bag(C)$ for short) the set of multi-sets over C and we represent a multi-set by the formal sum $a = \sum_{y \in C} a(y).y$. If a and b are two multi-sets over C, then $a + b$ is the multi-set over C defined by $a + b = \sum_{y \in C}(a(y) + b(y)).y$ and if λ is a natural, then $\lambda.a$ is the multi-set over C defined by $\lambda.a = \sum_{y \in C}(\lambda.a(x)).x$. One say that a is greater or equal than b, denoted $a \geq b$ if and only if $\forall y \in C, a(y) \geq b(y)$.*

Definition 2. *A colored net is a 6-tuple $CN = \langle P, T, \mathcal{C}, G, W^+, W^- \rangle$ with :*

- *P is the non empty and finite set of places*
- *T is the non empty and finite set of transitions (disjoint with P);*
- *$\mathcal{C}$ is the color mapping from $P \bigcup T$ to ω where ω is a set including the finite and non empty sets. An item of $\mathcal{C}(s)$ is called a color of s and $\mathcal{C}(s)$ denotes the color domain of s.*
- *G associates each transition t with a boolean application G_t on $\mathcal{C}(t)$ called the guard of t.*

– *W^+ (resp. W^-) is the post (resp. pre) incidence mapping that associates to each transition t and to each place p a color mapping from $\mathcal{C}(t)$ to $Bag(\mathcal{C}(p))$ which defines the tokens that are needed, consumed or produced by the firing of a transition (see def. 3). We note $W = W^+ - W^-$ (note that $W(t,p)$ is a mapping from $\mathcal{C}(t)$ to $Bag_{\mathbb{Z}}(\mathcal{C}(p))$).*

We note $\epsilon = \{\bullet\}$ the color domain reduced to the unique value $\bullet$ (the token); this allows us to consider ordinary Petri nets as particular colored Petri nets (the unique and common color domain is ϵ).

Definition 3 (Marking and Firing rule). *A marking is a mapping that associates to each place p a value in $Bag(\mathcal{C}(p))$. We note m_0 the initial marking of a net.*

A transition t is enabled for an instance $c_t \in \mathcal{C}(t)$ from a marking m (denoted by $m[t, c_t\rangle$) if

$$G_t(c_t) = True \text{ and } \forall p \in P, m(p) \geq W^-(t,p)(c_t)$$

The firing of t for $c_t \in \mathcal{C}(t)$ from m leads to the marking m' ($m[t,c_t\rangle m'$) defined by

$$\forall p \in P, m'(p) = m(p) + W^+(t,p)(c_t) - W^-(t,p)(c_t)$$

A marking m' is reachable from a marking m if there exists a sequence $t_1, c_1, \ldots, t_k, c_k$ such that $m[t_1,c_1\rangle m_1$, $m_1[t_2,c_2\rangle m_2$, …, $m_{k-1}[t_k,c_k\rangle m'$. We denote by $Acc(CN, m_0)$ the set of all reachable markings from m_0.

Consider the following colored net that models a solution to the dining philosophers paradigm. In this model, `X` denotes the identity mapping over the finite set D

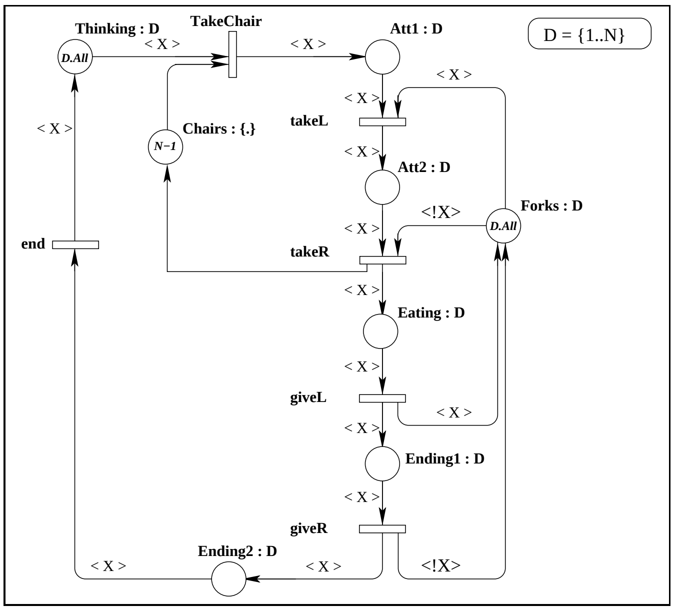

Fig. 1. A colored net for the dining philosophers paradigm

and `!X` denotes the "successor" mapping over D (i.e. the mapping that associates to an item x its successor in D that is supposed to be ordered). Semantic of this models is quite simple : a philosopher x who wants to eat must first take a chair by firing transition `takeChair`. Then it has to take in sequence its fork (transition *takeL*) and the fork of its right neightbour (transition *takeR*) to access state (place) `Eating`; the chair is released as soon as a philosopher gets its two forks (transition *takeR*). An eating philosopher can go back to the `Thinking` state by firing in sequence transitions `giveL`, `giveR` and `end`. Possible concurrent defects highlighted by this paradigm are the deadlock and the starvation problems.

Many analysis techniques have been adapted to colored nets, and in particular, automatic places invariants computation [19, 24, 20, 25, 26, 16, 7, 6]. Indeed, places invariants give rich informations on the behavior of the model without needing its execution; their definition and their computation involve only the structure of the model. Within different places invariants, positive flows are those that give the most usable information; this can be explained by the fact that the positive constraint added on weights simplify interpretation of these invariants. We recall now the definition of these places invariants.

Definition 4 (Colored positive flow). *Let C_{inv} be a color domain. A positive flow $\mathcal{F}$, with color domain C_{inv} ($\mathcal{C}(\mathcal{F}) = C_{inv}$), is a vector over P, noted as the formal sum $\mathcal{F} = \sum_{p \in P} \mathcal{F}_p.p$, such that $\forall p \in P, \mathcal{F}_p$ is a mapping from $Bag(\mathcal{C}(p))$ to $Bag(C_{inv})$ and such that $\forall t \in T$, $\sum_{p\in P} \mathcal{F}_p \circ W(t,p) = 0$* [1].

This definition implies that for any positive flow $\mathcal{F}$, and for any reachable marking m we have that $\sum_{p\in P} \mathcal{F}_p(m(p)) = \sum_{p \in P} \mathcal{F}_p(m_0(p))$. A flow can then be interpreted as an equations set linking the marking of a subset of places with the original marking of these places :

$$\forall m \in Acc(CN, m_0), \forall c \in \mathcal{C}(\mathcal{F}), \sum_{p\in P} \sum_{c_p \in \mathcal{C}(p)} [\mathcal{F}_p(c_p)(c)].m(p)(c_p) = cst(\in \mathbb{N})$$

Consider again the model of figure 1. In this net there are at least three positive flows on the color domain D:

- $\mathcal{F}_1 = \langle X\rangle.Thinking + \langle X\rangle.Att1 + \langle X\rangle.Att2 + \langle X\rangle.Eating + \langle X\rangle.Ending1 + \langle X\rangle.Ending2$
- $\mathcal{F}_2 = \langle !X\rangle.Forks + \langle !X\rangle.Att2 + \langle !X\rangle.Eating + \langle X\rangle.Eating + \langle X\rangle.Ending1$
- $\mathcal{F}_3 = \langle .\rangle.Att1 + \langle .\rangle.Att2 + \langle 1\rangle Chairs$

where $\langle X\rangle$ denotes the identity mapping over $Bag(D)$, $\langle !X\rangle$ denotes the successor mapping over $Bag(D)$, $\langle 1\rangle$ denotes the identity mapping over ϵ, and $\langle .\rangle$ the projection from D to *epsilon* defined by $\forall d \in D$, $.(d) = \bullet$. Note that by sake of simplicity we often do not note $\langle 1\rangle$.

The first one characterizes the sequential structure of philosophers that can be in one of the six states `Thinking`, `Att1`, `Att2`, `Eating`, `Ending1`, `Ending2`. Indeed, its interpretation is [2] : $\forall m \in Acc(CN, m_0), \forall x \in D, m(Thinking)(x) + m(Att1)(x) +$

[1] 0 denotes here the null mapping from $\mathcal{C}(t)$ to $Bag(\mathcal{C}(\mathcal{F}))$

[2] Remember that $\langle X\rangle(c)(c') = 1$ if $c = c'$ and $\langle X\rangle(c)(c') = 0$ otherwise.

$m(Att2)(x) + m(Eating)(x) + m(Ending1)(x) + m(Ending2)(x) = 1$. The second one, $\mathcal{F}_2$, tells us that, given $x \in D$, the fork $!x$ is either free (place `Forks` is marked with $!x$) or is either owned by philosopher $!x$ that is in state `Eating` or `Att2` or is used by philosopher x which is in state `Eating` or `Ending`.

At last, $\mathcal{F}_3$ highlights that chairs are either free (place `Chairs` is marked) or are shared by philosophers that are in state `Att1` or `Att2`. Its interpretation is $\forall m \in Acc(CN, m_0), \sum_{x \in C} m(Att1)(x) + \sum_{x \in C} m(Att2)(x) + m(Chairs) = N - 1$ and, combined with $\mathcal{F}_2$, ensures that place `Forks` cannot become empty and then, that no deadlock is possible.

As we can note, positive flows give precious information of the behavior of a model using only its structure.

However, computing positive flows is a difficult task. Up to now, only one algorithm exists [5] and is restricted to regular net (a sub-class of colored nets) with an unique color domain. A possible explanation is that most researches focus on a generative family computation (a family that generate all positive invariants of the net). This leads to very complex equations systems even if algorithms are defined on strong restriction of colored nets. We do not propose here to focus on a generative family but on **useful** family of positive flows. For doing this we impose a syntactic restriction on positive flows definition and we associate this restriction to a similar restriction on colored net definition. We call this kind of net "Simple Well-Formed nets" and we designed our positive flow as "simple positive flows".

2.2 Simple well-formed colored nets and simple positive flows

Modeling and verification are two strongly linked activities. A formalism has to define a good compromise between the simplicity provided for modeling and the richness of possible automatic tools or techniques that can be used to verify properties on the model.

In the Petri net domain, general formalisms have been proposed (like colored nets) but most of theoretical analysis results have been obtained on restrictions of general models. One can cite the regular nets [16], the ordered nets [8], and a formalism with a same modeling power as colored nets but with syntactic restrictions, the well-formed nets [3].

We propose here a slight restriction of this last formalism that we named simple well-formed nets (or SWF nets for short). This formalism remains sufficient for modeling large and complex problems; in particular we use it to model a large subset of Ada programs in order to validate them (see for instance models proposed in [12] or in [11]). As for the original model, we define our formalism by restricting the possible color mapping and the color domain construction.

Definition 5 (Basic color mapping). *Let C be a finite ordered set. The basic mapping are the* identity*, denoted by X_C, the* diffusion *(also called global synchronization), denoted by All_C, the* successor*, denoted by $!X_C$ and all* constant *mappings, $\lambda^c_C, c \in C$. They are defined from C to $Bag(C)$ by: $\forall x \in C$, $X_C(x) = x$, $All_C(x) = \sum_{c \in C} c$, $!X_C(x) =$ successor of x in C and $\lambda^c_C(x) = c$.*

Remark 1. It is a common usage to use other literals, $X, Y, Z, Ph, \ldots$, to denote the identity or successor mapping, and when the context is clear, one often omits the domain on which operates the mapping (X instead of X_C). All classes are considered to be ordered (in a circular way). So each item has a unique successor in the class (the successor of the last item is the first one). All these mappings can be extended to mapping from $Bag(C)$ to $Bag(C)$. When $C = \epsilon$ all these mappings coincide.

Definition 6 (Simple color mappings). *Let C be a finite ordered set. A simple color mapping on C is a mapping from C to $Bag(C)$ if either it's a constant mapping or if it can be written as a additive composition $\alpha.X_C + \beta.All_C + \gamma.!X_C$ with α, β, γ integer values such that $\beta \geq 0$, $\beta + \alpha \geq 0$ and $\beta + \gamma \geq 0$ (and $\beta + \alpha + \gamma \geq 0$ in the very case where $|C| = 1$).*

Remark 2. The constraints on α, β and γ ensure that a color mapping defines a positive value for each color of C (and then belongs to $Bag(C)$). When $C = \epsilon$ all simple color mappings are reduced to a constant mapping (an integer value).

Definition 7 (Simple color functions). *Let $C = C_1 \times C_2 \times \ldots \times C_k$ be a finite product of finite and non empty sets. A mapping f from C To $Bag(C)$ is a* simple color function *if it can be written $f = \langle f_1, f_2, \ldots, f_k \rangle$ with $\forall i$, f_i a simple color mapping on C_i or an arbitrary unitary [3] application from C_i to $Bag(C_i)$. If $\langle c_1, \ldots, c_k \rangle \in C$ then $f(c) = \langle f_1, f_2, \ldots, f_k \rangle(\langle c_1, \ldots, c_k \rangle) = \langle f_1(c_1), f_2(c_2), \ldots, f_k(c_k) \rangle$.*

When useful, we will note $\langle f_1, f_2, \ldots, f_k \rangle$ as $f_1.\langle f_2, \ldots, f_k \rangle$ and extend by linearity this notation to weighted sums of tuples.

We are in position to define nets we use to analyze Ada programs in our tool Quasar.

Definition 8 (Simple Well-Formed nets). *A colored net $\langle P, T, \mathcal{C}, G, W^-, W^+ \rangle$ is a* **simple well-formed net** *if $\forall p \in P$, $\forall t \in T$,*

- *$W^+(t,p) \neq 0$ or $W^-(t,p) \neq 0$ implies that $\mathcal{C}(t) = \mathcal{C}(p) \times C'_p$ ($\mathcal{C}(t)$ equals or includes $\mathcal{C}(p)$);*
- *$W^+(t,p)$ and $W^-(t,p)$ are a composition of a simple color function over $\mathcal{C}(p)$ with a projection from $\mathcal{C}(t)$ to $\mathcal{C}(p)$ (when color domain of t is "larger" than the color domain of p);*
- *if $W^+(t,p)$ uses a constant mapping on a class C_i then an arc between p and an other transition cannot use the mapping All_{C_i}.*

We say that the net is **homogeneous** *if all color domains are identical (i.e. $\exists C_1$, $\ldots$, C_K such that $\forall s \in P \cup T$, $\mathcal{C}(s) = C_1 \times \ldots \times C_k$), if all guards are always evaluated to True (there is no guards) and if all color mappings are only built with X, All and $!X$ basic mappings (no constants and no arbitrary mappings must appear on arcs).*

Remark 3. The third point is used to ensure that we can always homogenize a simple well-formed net; i.e. construct an equivalent model (or that perform a weak simulation of the original one) but with a unique color domain for each place and each transition (see subsection 3.5).

[3] f_i is an unitary application if $\forall c \in C_i, |f_i(c)| = 1$

SWF nets are a restriction of well-formed nets [3]. In particular, we do not allow guarded functions or additive composition of different instances of the same color class (e.g. $\langle X\rangle + \langle Y\rangle$ is forbidden). However, the expressiveness provided by this definition remains sufficient for modeling almost all problems. For instance, as said previously, we use this formalism to model precisely the behavioral semantics of Ada programs (with possible dynamics task creation) in order to analyze them.

We give now the definition of positive flows we will compute on SWF nets. This definition restricts the "functional" structure of the flow to a regular one.

Definition 9 (Simple positive flows). *A positive flow $\mathcal{F}$ on the color domain D is said to be a simple positive flow if $\forall p \in P$, $\mathcal{C}(p) = D \times \mathcal{C}'_p$ ($\mathcal{C}(p)$ "includes" D), $\mathcal{F}_p$ is a composition of a simple color function over $\mathcal{C}(p)$ using no constant with a projection from D to $\mathcal{C}(p)$.*

These restrictions are not very severe; indeed, the third last flows, presented with model of figure 1 are simple positive flows. Furthermore, this definition provides two advantages:

1. their definition uses only simple color mapping, and then, these positive flows can be very easily interpreted or used by specific tools (like structural reductions): they can characterize critical section (like $\mathcal{F}3$ in the previous example), they can also characterize process structure ($\mathcal{F}1$) or the way resources are shared ($\mathcal{F}2$);
2. they can be computed in a systematic way as we will see in the next section.

For more clarity, we use in positive flow notation a dot . to highlight the projection from $\mathcal{C}(p)$ to the color domain of the flow. For instance, if the color domain of a flow is $D = C_1 \times C_3$ and the color domain of a place involved in the flow is $C = C_1 \times C_2 \times C_1 \times C_3$, we will note $\langle ., ., X', Z\rangle$ to denote the composition of the mapping $\langle X, Y, X', Z\rangle$ with the projection Π from C to D defined by $\Pi(x, y, x', z) = (x', z)$.

Given an homogeneous SWF net we can construct from its incidence matrix W the integer matrix $W^{n_1,\dots,n_k}$, indexed by $(T \times C1 \times \ldots \times C_k) \times (P \times C1 \times \ldots \times C_k)$ and defined by:

$$W^{n_1,\dots,n_k}(t, c'_1, \dots, c'_k)(p, c_1, \dots, c_k) = W(t,p)(\langle c'_1, \dots, c'_k\rangle)(\langle c1, \dots, c_k\rangle)$$

This construction consists only in "unfolding" color mapping that constitutes coefficients of the matrix W. In the same way, it is possible to "unfold" a positive flow $\mathcal{F}$ by building the set of integer vectors defined by all possible interpretations of the colored flow. Indeed, given a positive flow $\mathcal{F}$ and an interpretation c_{inv} we can define the integer vector $\overrightarrow{F}_{c_{inv}}$ indexed by $(C1 \times \ldots \times C_k \times P)$ and defined by:

$$\overrightarrow{F}_{c_{inv}}(c_1, \dots, c_k, p) = \mathcal{F}_p(\langle c1, \dots, c_k\rangle)(c_{inv})$$

Using these notations we have by definition:

Proposition 1. *$\mathcal{F}$ is a positive flow if and only if $\forall c_{inv} \in C$, $W^{n_1,\dots,n_k}.\overrightarrow{F}_{c_{inv}} = 0$.*

So, computing positive flows of a homogeneous colored net consists in solving the system $W^{n_1,\dots,n_k}.\overrightarrow{F}_{c_{inv}} = 0$ with, for instance, the Farkas algorithm described in [4]. However, this raises two main drawbacks:

1. this calculation requires to fix the parameters which is then equivalent to unfold the net and to compute positive flows in the unfolded net that is very inefficient;
2. as computed flows are integer vectors, it is very difficult to "recolor" them in the general case and leads to useless invariants.

We will prove that it is possible to solve this system in a **parametric way** without unfolding the net. For doing that we first prove that incidence matrix can be reordered in a “fractal” form.

3 Simple positive flows computation for SWF nets

In this section we propose to show how to compute a generative family of simple positive flows in a SWF net. Note that the computed set generates all simple positive flows **but not all** positive flows.

For this purpose, we first suppose that every considered net is homogeneous (we provide, in the last subsection, the mechanism used to transform a simple well-formed net into an homogeneous one). We show then that the constraints on SWF nets mappings and on simple positive flows definitions lead to a system with a “fractal” form which can be reduce to a set of non parametric equations.

We adopt the following notations :

- $C = C_1 \times C_2 \times \ldots \times C_k$ denotes the common color domain of places and transitions;
- $n_1 = |C_1|$, $n_2 = |C_2|$, ..., $n_k = |C_k|$,
- $C_j = \{c_j^i\}_{i=1..n_j}$;
- $\forall p \in P, \forall t \in T$, $W(t,p) = \langle w_1(t,p), \ldots, w_k(t,p)\rangle$ with for all i in $[1..k]$, $w_i(t,p) = a_i(t,p).X_{C_i} + b_i(t,p).All_{C_i} + d_i(t,p).!X_{C_i}$
- we compute positive flow on domain C; so we fix $C_{inv} = C$.
- if $\mathcal{F} = \sum_p \mathcal{F}_p.p$ is a simple positive flow, we note $\mathcal{F}_p = \langle f_1^p, \ldots, f_k^p\rangle$ and $f_i^p = \alpha_i.X_i + \beta_i.All_i + \gamma_i.(!X_i)$;
- we note $E = (\mathbb{Q}^+)^P$;

3.1 Reordering equations

The matrix of an homogeneous SWF net can be defined by a recursive construction, highlighting a "fractal" form that can be used to define an efficient algorithm for simple positive flow computation. In this construction, transitions are organized into lines (corresponding to the equations of the system) and places into columns (corresponding to the variables of the system). This construction is based on the notion of block matrix that we recall now.

Definition 10 (Square block matrix). *A matrix $A = (a_{i,j})$ in $\mathbb{N}^{K.n \times K.m}$ is an integer square block matrix if each $a_{i,j}$ is a $m \times n$ integer matrix or a square block matrix (in which case, $n = k'.n'$ and $m = k'.m'$). We note $A(i,j) = a_{i,j}$ the item on i^{th} line and j^{th} column.*

Definition 11 (Matrix fractal form). *A block matrix $W = (w_{i,j})$ in $\mathbb{N}^{K.n \times K.m}$ has a "Simple Well Formed Net Fractal form" (or has a fractal form for short) if there exist three matrices A, B, D with the same dimension such that :*

1. *matrices A, B, D are either three integer matrices or three block matrices with also a fractal form;*
2. *items of W satisfy:*

$$\forall i, j \in 1..K, w_{i,j} = \begin{cases} A + B & \text{if } i{=}j \\ D + B & \text{if } j{=}i{+}1 \text{ modulo } n \\ B & \text{in other cases} \end{cases}$$

For instance, if A, B, D are three integer matrices with the same dimension then the following $n \times n$ matrix has a fractal form.

$$W = \begin{bmatrix} (A+B) & (D+B) & B & \dots & B \\ B & (A+B) & (D+B) & \dots & B \\ \dots & \dots & \ddots & \ddots & \dots \\ B & \dots & B & (A+B) & (D+B) \\ (D+B) & B & \dots & B & (A+B) \end{bmatrix}$$

Consider now an homogeneous SWF net (W is its incidence matrix). Remember that k denotes the number of classes of the net and then, the number of different parameters of the system and that $W(t,p) = \langle w_1(t,p), \dots, w_k(t,p) \rangle$ with for all i in $[1..k]$, $w_i(t,p) = a_i(t,p).X_{C_i} + b_i(t,p).All_{C_i} + d_i(t,p).!X_{C_i}$.

Definition 12 (Extracting a fractal form of a homogeneous SWF net). *Given $v \in [1..k+1]$, and three sets $I_A, I_B, I_D \subseteq \{n_1, n_2, \dots, n_k\}$ we define the integer or block matrix $W_v^{I_A,I_B,I_D}$ recursively by:*

– *if $v = k+1$ then $W_{k+1}^{I_A,I_B,I_D} = (w_{t,p})$ is the $T \times P$ integer matrix*

$$w_{t,p} = \prod_{i \in I_A} a_i(t,p). \prod_{j \in I_D} d_j(t,p). \prod_{l \in I_B} b_l(t,p) \quad [4]$$

– *if $v \leq k$, then $W_v^{I_A,I_B,I_D} = (w_{i,j})$ is the $n_v \times n_v$ square block matrix defined by*

$$w_{i,j} = \begin{bmatrix} W_{v+1}^{I_A \cup \{v\}, I_B, I_D} + W_{v+1}^{I_A, I_B \cup \{v\}, I_D} & \text{if } i{=}j \\ W_{v+1}^{I_A, I_B, I_D \cup \{v\}} + W_{v+1}^{I_A, I_B \cup \{v\}, I_D} & \text{if } j{=}i{+}1 \text{ modulo } n_v \\ W_{v+1}^{I_A, I_B \cup \{v\}, I_D} & \text{in other cases} \end{bmatrix}$$

We note W' the square block matrix $W_1^{\emptyset,\emptyset,\emptyset}$ and when there is no ambiguity, we note $A = W_2^{\{1\},\emptyset,\emptyset}$, $B = W_2^{\emptyset,\{1\},\emptyset}$ and $D = W_2^{\emptyset,\emptyset,\{1\}}$.

Proposition 2. *We have the following results:*

1. *The two matrices W and W' are equivalent for defining a SWF homogenous net; i.e. $\forall p \in P, \forall t \in T$, $\forall c = \langle c_1, \dots, c_k \rangle \in C$, $\forall c' = \langle c'_1, \dots, c'_k \rangle \in C$, we have $W'(c_1, c'1)(c_2, c'2) \dots (c_k, c'k)(t,p) = W(t,p)(c')(c)$.*
2. *The matrix W' has a fractal form (as soon as $k \geq 1$).*

[4] In this product we use the convention that a product on the empty set equals 1 ($\prod_{i \in \emptyset} f(i) = 1$)

Proof. Point 2 is a direct consequence of the definition of $W_1^{\emptyset,\emptyset,\emptyset}$. For proving point 1, it sufficient to note that $W(t,p)(\langle c_1,\ldots,c_k\rangle) = \langle w_1(t,p)(c_1),\ldots,w_k(t,p)(c_k)\rangle$ and that $w_i(t,p)(c_i)(c'_i) = b_i(t,p) + \delta_{c_i,c'_i}.a_i(t,p) + \delta_{c_i,!c'_i}.d_i(t,p)$ where δ is the Kronecker symbol defined by $\delta_{c_i,c'_i} = 0$ if $c_i \neq c'_i$ and $\delta_{c_i,c_i} = 1$; it comes that

$$W(t,p)(\langle c_1,\ldots,c_k\rangle)(\langle c'_1,\ldots,c'_k\rangle) = \Pi_{i=1..k}(b_i(t,p) + \delta_{c_i,c'_i}.a_i(t,p) + \delta_{c_i,!c'_i}.d_i(t,p))$$

Now remark that when $c_v = c'_v$ we have added v to the set I_A, when $c_v = !c'_v$ we have added v to the set I_D and that we use these sets to compute the integer values of the latest matrices and we obtain the result.

$\square$

If we consider an homogeneous SWF net with two classes (two parameters), the incidence matrix can be written:

$$W = \begin{bmatrix} (A+B) & (D+B) & B & \ldots & B \\ B & (A+B) & (D+B) & \ldots & B \\ & & \ddots & \ddots & \\ \ldots & \ldots & & & \ldots \\ B & \ldots & B & (A+B) & (D+B) \\ (D+B) & B & \ldots & B & (A+B) \end{bmatrix}$$

with

$$A = W_2^{\{1\},\emptyset,\emptyset} = \begin{bmatrix} (AA+AB) & (AD+AB) & AB & \ldots & AB \\ AB & (AA+AB) & (AD+AB) & \ldots & AB \\ & & \ddots & \ddots & \\ \ldots & \ldots & & & \ldots \\ AB & \ldots & AB & (AA+AB) & (AD+AB) \\ (AD+AB) & AB & \ldots & AB & (AA+AB) \end{bmatrix}$$

$$B = W_2^{\emptyset,\{1\},\emptyset} = \begin{bmatrix} (BA+BB) & (BD+BB) & BB & \ldots & BB \\ BB & (BA+BB) & (BD+BB) & \ldots & BB \\ & & \ddots & \ddots & \\ \ldots & \ldots & & & \ldots \\ BB & \ldots & BB & (BA+BB) & (BD+BB) \\ (BD+BB) & BB & \ldots & BB & (BA+BB) \end{bmatrix}$$

$$D = W_2^{\emptyset,\emptyset,\{1\}} = \begin{bmatrix} (DA+DB) & (DD+DB) & DB & \ldots & DB \\ DB & (DA+DB) & (DD+DB) & \ldots & DB \\ & & \ddots & \ddots & \\ \ldots & \ldots & & & \ldots \\ DB & \ldots & DB & (DA+DB) & (DD+DB) \\ (DD+DB) & DB & \ldots & DB & (DA+DB) \end{bmatrix}$$

and with

- $AA(t,p) = a_1(t,p).a_2(t,p)$, $AB(t,p) = a_1(t,p).b_2(t,p)$, $AD(t,p) = a_1(t,p).d_2(t,p)$
- $BA(t,p) = b_1(t,p).a_2(t,p)$, $BB(t,p) = b_1(t,p).b_2(t,p)$, $BD(t,p) = b_1(t,p).d_2(t,p)$
- $DA(t,p) = d_1(t,p).a_2(t,p)$, $DB(t,p) = d_1(t,p).b_2(t,p)$, $DD(t,p) = d_1(t,p).d_2(t,p)$

3.2 Reordering solutions and simplifying equations

First, note that any simple positive flow $\mathcal{F}$ can be written in a unique way as a sum $F_{<>} = \langle X_1, X_2, \ldots, X_k\rangle f_1 + \langle X_1, X_2, \ldots, !X_k\rangle f_2 + \langle X_1, X_2, \ldots, All_k\rangle f_3 + \ldots + \langle All_1, All_2, \ldots, All_k\rangle f_{3^k}$ with f_i integer vectors over P.

Second, remark that the reorganization performed on the incidence matrix can also be applied to the solutions of the studied system (and we need to do it).

Indeed, a positive flow $\mathcal{F}$ (in a functional form) defines for each value $c_{inv} \in C$ a developed vector $\overrightarrow{F}_{c_{inv}}$ in $E^{n_1 \times \ldots \times n_k}$. Remark also that this vector can be viewed as a vector of $(E^{n_2 \times \ldots \times n_k})^{n_1}$ i.e. a vector of size n_1 with each component in $E^{n_2 \times \ldots \times n_k}$.

$$\overrightarrow{F}_{c_{inv}} = \begin{bmatrix} \overrightarrow{F[1]}_{c_{inv}} \\ \vdots \\ \overrightarrow{F[n_1]}_{c_{inv}} \end{bmatrix} \text{ with } \forall i \in 1..n_1, \overrightarrow{F[i]}_{c_{inv}}(c_2, \ldots, c_k, p) = \overrightarrow{F}_{c_{inv}}(c_1^i, c_2, \ldots, c_k, p)$$

As we restrict positive flow computation to simple positive flow computation we can use the particular form of such vector and write them in a "parametric" form.

Proposition 3. *Given a* **simple** *positive flow $\mathcal{F}$ and a color interpretation c_{inv}, then $\overrightarrow{F}_{c_{inv}}$ has a unique decomposition :*

$$\overrightarrow{F}_{c_{inv}} = \begin{bmatrix} 0 \\ \ldots \\ 0 \\ F_X \\ 0 \\ \ldots \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \ldots \\ 0 \\ 0 \\ F_{!X} \\ \ldots \\ 0 \end{bmatrix} + \begin{bmatrix} F_{All} \\ \ldots \\ F_{All} \\ F_{All} \\ F_{All} \\ \ldots \\ F_{All} \end{bmatrix}$$

where F_X, $F_{!X}$ and F_{All} are three $E^{n_2 \times \ldots \times n_k}$ vectors depending on the value of $\{\alpha_i^p\}_p$ for F_X, of $\{\gamma_i^p\}_p$ for $F_{!X}$ and of $\{\alpha_i^p, \gamma_i^p, \beta_i^p\}_p$ for F_{All}, and such that if, $c_{inv} = \langle c_1^i, c_2, \ldots, c_k \rangle$, then F_X is on the i^{th} row and $F_{!X}$ is on the $(i+1)^{th}$ row.

Proof. Let $c_{inv} = \langle c_1'^i, c_2', \ldots, c_k' \rangle$ and $j, j' \in 1..n_1$ both distinct of i and $i+1$. Suppose that $\overrightarrow{V}$ is the vector defined by $\overrightarrow{V} = \overrightarrow{F[j]}_{c_{inv}} - \overrightarrow{F[j']}_{c_{inv}}$. We have $\forall p, c_2, \ldots, c_k$, $\overrightarrow{V}(p, c_2, \ldots, c_k) = \overrightarrow{F}_{c_{inv}}(p, c_1^j, c_2, \ldots, c_k) - \overrightarrow{F}_{c_{inv}}(p, c_1^{j'}, c_2, \ldots, c_k)$.

So, $\overrightarrow{V}(p, c_2, \ldots, c_k) = \mathcal{F}_p(\langle c_1^j, \ldots, c_k \rangle)(c_{inv}) - \mathcal{F}_p(\langle c_1^{j'}, \ldots, c_k \rangle)(c_{inv})$ that can be written $\langle f_1^p(c_1^j, c_1^i), f_2^p(c_2, c'2), \ldots, f_k^p(c_k, c'k) \rangle - \langle f_1^p(c_1^{j'}, c_1^i), f_2^p(c_2, c'2), \ldots, f_k^p(c_k, c'k) \rangle$. As j and j' are **both** distinct of i and $i+1$ it comes that $f_1^p(c_1^j, c_1^i) - f_1^p(c_1^{j'}, c_1^i) = 0$ and then $\overrightarrow{V} = \overrightarrow{0}$. We note F_{All} the vector $\overrightarrow{F[j]}_{c_{inv}}$.

Let then $\overrightarrow{V_X}$ the vector defined by $\overrightarrow{V_X} = \overrightarrow{F[i]}_{c_{inv}} - \overrightarrow{F[j]}_{c_{inv}}$. Using a same argumentation it comes that $\overrightarrow{V_X}(p, c_2, \ldots, c_k) = \langle f_1^p(c_1^i, c_1^i), f_2^p(c_2, c'2), \ldots, f_k^p(c_k, c'k) \rangle - \langle f_1^p(c_1^j, c_1^i), f_2^p(c_2, c'2), \ldots, f_k^p(c_k, c'k) \rangle$. As $f_1^p(c_1^i, c_1^i) - f_1^p(c_1^j, c_1^i) = \alpha_i^p$ we can note $F_X = V_X$. We can proceed also to the same construction for defining in an unique way $F_{!X} = V_{!X}$.

□ □

Now, if we combine the regular fractal form of a SWF homogeneous net with the particular form of simple positive flows, we can simplify the system that has to be solved.

Proposition 4. *Using previous notations we have that $W_{n_1, \ldots, n_k}.\overrightarrow{F}_{c_{inv}} = 0$ iff:*

$$B.F_{All} = A.F_{!X} = D.F_X = A.F_X + D.F_{!X} = B.(F_X + F_{!X}) + (A + D).F_{All} = 0$$

Proof. The system can be written

$$\begin{bmatrix} (A+B) & (D+B) & B & \dots & B \\ B & (A+B) & (D+B) & \dots & B \\ \dots & \dots & \ddots & \ddots & \dots \\ B & \dots & B & (A+B) & (D+B) \\ (D+B) & B & \dots & B & (A+B) \end{bmatrix} . \begin{bmatrix} F_{All} \\ \dots \\ F_{All} \\ F_X + F_{All} \\ F_{!X} + F_{All} \\ \dots \\ F_{All} \end{bmatrix} = 0$$

Since simple positive flows are defined only with integer vectors (they don't use n_1 as coefficient) and since a simple positive flow defines solutions for any value of n_1, it comes that $B.F_{All} = 0$. If we develop now equations (and using the fact that $B.F_{All} = 0$), we obtain only four distinct equations:

- $(A + D).F_{All} + B.(F_X + F_{!X}) = 0$;
- $(A + B).F_X + (B + D).F_{!X} + (A + D).F_{All} = 0$;
- $B.F_X + (A + B).F_{!X} + (A + D).F_{All} = 0$;
- $(B + D).F_X + B.F_{!X} + (A + D).F_All = 0$.

By subtracting the first one to the others we obtain the result. Now, if F_X, F_{All} and $F_{!X}$ fulfill the previous equation it is clear that the vector $\overrightarrow{F}_{c_{inv}}$ is solution of $\overrightarrow{F}_{c_{inv}} = 0$ (whatever the positive value of n_1 is).

□ □

3.3 Computing simple positive flow in the homogenous case

We are now in position to propose an algorithm for computing simple positive flow for an homogeneous SWF net. For doing that we need to define two matrices operators: the first one, the "stacking" operator define how to stack matrices of the same dimension. The second one, the "juxtaposition" operator, define how to put side by side matrices of the same dimension. These two operators differ from classical ones in the sense that they keep the fractal structure of matrices when stacking or juxtaposing them.

Definition 13 (Stacking and juxtaposing matrices). *Let $W^1 = [w^1_{i,j}]_{i \in [1..n], j \in [1..m]}$, $\dots$, $W^q = [w^q_{i,j}]_{i \in [1..n], j \in [1..m]}$ q matrices.*

1. *If $W^1, \dots, W^q$ are all integer matrices then*
   - *the stacking of $W^1, \dots, W^q$, noted $\left[W^1/\dots/W^q\right]$, is the matrix $[s_{i,j}]_{i \in [1..q.n], j \in [1..m]}$ (m columns and $q.n$ lines) with $s_{(q.i)-r,j} = w^{q-r}_{i,j}$, $r \in 0..q-1$.*
   - *the juxtaposition of $W^1, \dots, W^q$, noted $\left[W^1|\dots|W^q\right]$, is the matrix $[s_{i,j}]_{i \in [1..n], j \in [1..q.m]}$ (n lines and $q.m$ columns) with $s_{i,(q.j)-r} = w^{q-r}_{i,j}$, $r \in 0..q-1$.*
2. *If $W^1, \dots, W^q$ are all block matrices (their items are others matrices) then*
   - *the stacking of $W^1, \dots, W^q$, noted $\left[W^1/\dots/W^q\right]$, is the matrix $[s_{i,j}]_{i,j \in [1..n]}$ recursively defined by $s_{i,j} = \left[w^1_{i,j}/\dots/w^q_{i,j}\right]$*
   - *the juxtaposition of $W^1, \dots, W^q$, noted $\left[W^1|\dots|W^q\right]$, is the matrix $[s_{i,j}]_{i,j \in [1..n]}$ recursively defined by $s_{i,j} = \left[w^1_{i,j}|\dots|w^q_{i,j}\right]$*

Remark 4. As a vector can be seen as a single column matrix, these two operators can also be applied to vectors.

Proposition 5. *If $W^1, \ldots, W^q$ are fractal matrices of the same dimension then $\left[W^1/\ldots/W^q\right]$ and $\left[W^1|\ldots|W^q\right]$ are also fractal matrices.*

Using these operators, we can rewrite previous system.

Proposition 6. *Using previous notations, we have that $W_{n1,\ldots,n_k}.\overrightarrow{F}_{c_{inv}} = 0$ iff:*

$$\left[\,[0|0|B]\,/\,[0|A|0]\,/\,[D|0|0]\,/\,[A|D|0]\,/\,[B|B|A+D]\,\right].\left[\,F_X/\ F_{!X}/\ F_{All}\,\right] = 0$$

Proof. A direct consequence of the operators definition.

□ □

We propose now an algorithm that computes a generative family of simple positive flows. Input of this algorithm is either an integer matrix and a set Parameters reduced to the empty set (no parameterized system) or a fractal block matrix with a set Parameters compatible with W (the size of W has the size $n_1 \times n_1$ and each items is either an integer matrix or a fractal one with the size $n_2 \times n_2$ and so on). The output is either a set of integer vectors (when Parameters is reduced to the empty set) or a set of formal sums $F_{<>} = \langle X_1, X_2, \ldots, X_k\rangle f_1 + \langle X_1, X_2, \ldots, !X_k\rangle f_2 + \langle X_1, X_2, \ldots, All_k\rangle f_3 + \ldots + \langle All_1, All_2, \ldots, All_k\rangle f_{3^k}$ with f_i integer vectors over P that generate simple positive flows.

Algorithm 1 : Simple_Positive_Solutions(W, Parameters = $\{n_1, \ldots, n_k\}$)

```
If (Parameters = $\emptyset$) Then
 + return $\{X|W.X = 0\}$ – *integer vectors computed with the Farkas algorithm*
Else
 + Construct the **fractal** matrix $W'$ defined by [5]

        $W' = \left[\,[0|0|B]\,/\,[0|A|0]\,/\,[D|0|0]\,/\,[A|D|0]\,/\,[B|B|A+D]\,\right]$

 + Compute the set $SF$ of solutions of the system $W'.\left[\,F_X/\ F_{!X}/\ F_{All}\,\right] = 0$.
   with this algorithm: $SF :=$ Simple_Positive_Flows(W',$\{n_2, \ldots, n_k\}$ ) [6]
 + Return the set of formal sums

        $\left\{F = X_{C_1}.F_X + All_{C_1}.F_{All} + !X_{C_1}.F_{!X}, \left[\,F_X/\ F_{!X}/\ F_{All}\,\right] \in SF\right\}$

End if;
```

Proposition 7. *The set computed by the previous algorithm defines a generative family of simple positive flows (of a SWF net).*

Proof. By recurrence on the set Parameters:

1. if Parameters = $\emptyset$ then the set computed is a generative family since we use the Farkas algorithm;

[5] as previously, we note A, B and D the blocks of the fractal matrix W

[6] if $k < 2$, then $\{n_2, \ldots, n_k\} = \emptyset$

2. Suppose that given any fractal matrix W' and a compatible set $\{n_2, \ldots, n_k\}$ the previous algorithm computes a generative family of simple positive flows.
   (a) formal sums computed by the algorithm define simple positive flows of net defined by W. Indeed, recurrence hypothesis combined with proposition 6 ensure that we effectively compute simple positive flows.
   (b) the set is generative. Indeed, let $\mathcal{F}0$ be a simple positive flow. Using proposition 3, any interpretation $\overrightarrow{F0}_{c_{inv}}$ of $\mathcal{F}0$ can be written with $F0_X$, $F0_{All}$ and $F0_{!X}$ as defined in this proposition. Using the proposition 6, we obtain that $W'.\left[F0_X/\ F0_{!X}/\ F0_{All}\right] = 0$. By recurrence hypothesis, as we compute a generative family of solutions of $W'.\left[F_X/\ F_{!X}/\ F_{All}\right] = 0$ then $\left[F0_X/\ F0_{!X}/\ F0_{All}\right]$ is generated by this set and then all interpretation of $\mathcal{F}0$ is generated by the set computed. So, formal sums computed by the previous algorithm generate all simple positive flows of the SWF net defining by the fractal matrix W.

$\square$ $\square$

If we note $K_{P\times T}$ the complexity of the Farkas algorithm for a net with P places and T transitions, then the complexity of the previous algorithm is $K_{3^k.P\times 5^k.T}$ with k the number of classes of the net. However, as matrices built by the algorithm are very sparce, first results we obtained seem to prove that the algorithm behaves as its complexity was $2^k.K_{P\times T}$ which is a good complexity since, even for very complex models, k remains lower than 5.

3.4 Dealing with non homogeneous SWF nets

Suppose now that the net is not homogeneous. In order to use previous algorithm, we have to homogenize the net. Two different cases have to be considered : 1) a transition has a color domain larger than its adjacent places (the contrary is not possible due to the definition of SWF nets); 2) a color mapping use a constant value or an arbitrary mapping. In order to make homogeneous the studied net we proceed in two steps :

1. as soon as a constant mapping (of a class C) or an arbitrary mapping appears on an arc valuation, we replace all mappings on this class by the mapping All_C; this replacement leads to a synchronization loosening and then the obtained net makes a weak simulation of the original one. So, all computed flows in this net are also flows of the original one [7]. Furthermore, as we forbid the mixing of constant and of the mapping All_C in the simple well-formed net definition, we have not to fix the parameter size of C (which would be necessary if we need to homogenize an arc with mapping All_C since it would be replaced by $|C|.All_C$).
2. compute the lowest common multiple (C_{lcm}) of color domain (by extending classical multiplication and division to product of classes) and extend color domain of each place (and of each transition) such that their color domain equals C_{lcm}. Modify also accordingly the original marking. For instance, the lcm of $C_1 \times C_1 \times C_2$, $C_1 \times C_2 \times C_3$ and $C3 \times C_3$ is $C_1 \times C_1 \times C2 \times C_3 \times C_3$. Suppress in each flow computed the additive color part. As these transformations do not modify the behavior of the model it is clear that computed positive flows by this manner are those of the original model.

[7] It is possible to treat cleverly constant mapping.

For instance, consider the simple well-formed net depicted in Figure 2 that models an atomic assignment `Free := f(Id, X)` where f is an arbitry Ada boolean function and where places `Write` models a read-write lock.

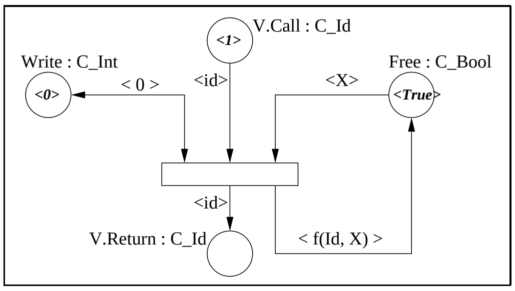

Fig. 2. A simple net

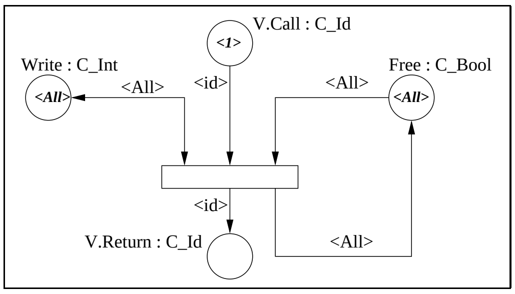

Fig. 3. The previous net after a first homogenization

The first step (constant and arbitrary mapping homogenization) produces the model depicted Figure 3. The lowest common multiple color domain is $C\_Id \times C\_Int \times C\_Bool$. After homogenization of the net we obtain the model depicted in Figure 4. On can remark that information concerning color domain C_Bool have been forgotten by the homogenization process (and that the model is quite less readable).

Once homogenization is done, one can compute positive flow : for instance the sum $\langle X, All, All\rangle.(V.Call + V.Return)$ defines a simple positive flow of the latest model. In order to obtain positive flows of the original model it's sufficient to suppress carefully color part added for homogenization : for the previous invariant, we obtain the "correct" positive flow $\langle X\rangle.(V.Call + V.Return)$.

3.5 Example

Consider again the net of figure 1. Its incidence matrix after homogenization is:

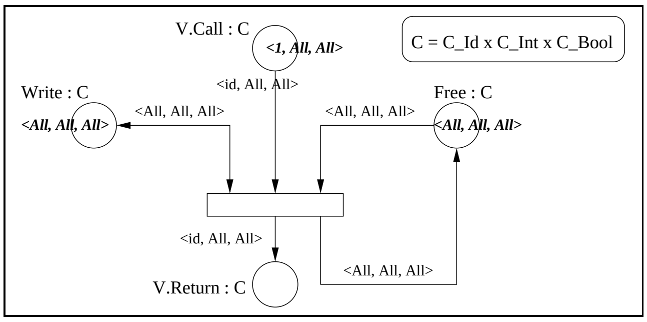

Fig. 4. The previous net homogenized

$$W = \begin{array}{l|cccccccc} & Thinking & Att1 & Att2 & Eating & Ending1 & Ending2 & Forks & Chairs \\ \hline TakeChairs & -\langle X\rangle & \langle X\rangle & 0 & 0 & 0 & 0 & 0 & -\langle All\rangle \\ takeL & 0 & -\langle X\rangle & \langle X\rangle & 0 & 0 & 0 & -\langle X\rangle & 0 \\ takeR & 0 & 0 & -\langle X\rangle & \langle X\rangle & 0 & 0 & -\langle !X\rangle & \langle All\rangle \\ giveL & 0 & 0 & 0 & -\langle X\rangle & \langle X\rangle & 0 & \langle X\rangle & 0 \\ giveR & 0 & 0 & 0 & 0 & -\langle X\rangle & \langle X\rangle & \langle !X\rangle & 0 \\ end & \langle X\rangle & 0 & 0 & 0 & 0 & -\langle X\rangle & 0 & 0 \end{array}$$

The corresponding matrices A, B and D are:

$$A = \begin{array}{l|cccccccc} & Thinking & Att1 & Att2 & Eating & Ending1 & Ending2 & Forks & Chairs \\ \hline TakeChairs & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ takeL & 0 & -1 & 1 & 0 & 0 & 0 & -1 & 0 \\ takeR & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ giveL & 0 & 0 & 0 & -1 & 1 & 0 & 1 & 0 \\ giveR & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ end & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \end{array}$$

$$B = \begin{array}{l|cccccccc} & Thinking & Att1 & Att2 & Eating & Ending1 & Ending2 & Forks & Chairs \\ \hline TakeChairs & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ takeL & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ takeR & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ giveL & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ giveR & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ end & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

$$D = \begin{array}{l|cccccccc} & Thinking & Att1 & Att2 & Eating & Ending1 & Ending2 & Forks & Chairs \\ \hline TakeChairs & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ takeL & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ takeR & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ giveL & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ giveR & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ end & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Applying algorithm of page 13 leads to 6 flows:

- $\mathcal{F}_1 = \langle X\rangle.Thinking + \langle X\rangle.Att1 + \langle X\rangle.Att2 + \langle X\rangle.Eating + \langle X\rangle.Ending1 + \langle X\rangle.Ending2$

- $\mathcal{F}'_1 = \langle !X\rangle.Thinking + \langle !X\rangle.Att1 + \langle !X\rangle.Att2 + \langle !X\rangle.Eating + \langle !X\rangle.Ending1 + \langle !X\rangle.Ending2$
- $\mathcal{F}''_1 = \langle All\rangle.Thinking + \langle All\rangle.Att1 + \langle All\rangle.Att2 + \langle All\rangle.Eating + \langle All\rangle.Ending1 + \langle All\rangle.Ending2$
- $\mathcal{F}_2 = \langle !X\rangle.Forks + \langle !X\rangle.Att2 + \langle !X\rangle.Eating + \langle X\rangle.Eating + \langle X\rangle.Ending1$
- $\mathcal{F}_3 = \langle All\rangle.Att1 + \langle All\rangle.Att2 + \langle X\rangle Chairs$ $\mathcal{F}'_3 = \langle All\rangle.Att1 + \langle All\rangle.Att2 + \langle X!\rangle Chairs$

This example emphasizes that, in many cases, our algorithm computes a generative family of all positive flows. However, it underlines also two difficulties associated to our method. First we compute useless flows such as $\mathcal{F}'_1$, $\mathcal{F}''_1$ or $\mathcal{F}'_3$; indeed, as soon as $\langle X_C\rangle.F$ is a flow $\langle X!_C\rangle.F$ and $\langle All\rangle.F$ are also two flows and if $\langle X_C\rangle.F + \langle X!_C\rangle.F'$ is a flow then $\langle All\rangle.(F + F')$ is also a flow. Our first experimentations show that we compute in average one useless flow per flow and per color domain. As the complexity of the Farkas algorithm depends principally on the number of solution, our method behaves as if we compute positive flows on a net two times bigger than the original one. We are studying algorithm heuristics to solve this slight problem. The second problem is that, by definition, some positive flows cannot be computed. For instance, a flow involving three different colors (X, $X!$ and $X!!$) or a flow using the cardinal of a class as weight ($n_1.\langle X\rangle.F$) are not simple positive flows and thus, are not computed. If we don't foresee now a solution to include parameters in flows definition, we can easily adapt the definition and the associated computation algorithm of simple positive flows to take into account more complex flows.

4 Conclusion

We have proposed an algorithm that computes a generative family of particular but useful positive flows of a slightly restricted subclass of colored nets. This algorithm is being implemented in our tool Helena [9] (http://helena.cnam.fr) and its distributed version Cyclades [22]. It will be used to enforce structural techniques, such as structural reductions, stubborn sets computation or distributed partitioning used in these tools to verify concurrent programs in the Quasar project (http://quasar.cnam.fr).

The way we define our algorithm allows its extension to other kinds of nets as soon as they provide some regularity. For instance, it can be immediately adapted to deal with all non guarded mappings used in normalized symmetric nets definitions[18].

Remark for the reviewers: the length of the paper may be easily reduced by compacting some matrices and/or suppressing proofs and replacing them by a citation of a technical report accessible on our web site.

References

1. G. Berthelot. Checking properties of nets using transformations. In G. Rozenberg, editor, *Advances in Petri nets*, volume No. 222 of *LNCS*. Springer-Verlag, 1985.
2. E. Bruneton and J.F. Pradat-Peyre. Automatic verification of concurrent ada programs. In Michael Gonzalez Harbour and Juan A. de la Puente, editors, *Reliable Software Technologies-Ada-Europe'99*, number 1622 in LNCS, pages 146–157. Springer-Verlag, 1999.

3. C. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. On well-formed colored nets and their symbolic reachability graph. In *ICATPN*, Paris-France, June 1990.
4. J. M. Colom and M. Silva. Convex geometry and semiflows in P/T nets. A comparative study of algorithms for computation of minimal P-semiflows. *Lecture Notes in Computer Science; Advances in Petri Nets 1990*, 483:79–112, 1991. NewsletterInfo: 33,39.
5. J. M. Couvreur, S. Haddad, and J. F. Peyre. Computation of generative families of positive semi-flows in two types of coloured nets. In *Proceedings of the 12th International Conference on Application and Theory of Petri Nets, 1991, Gjern, Denmark*, pages 122–144, June 1991. NewsletterInfo: 39.
6. J. M. Couvreur, S. Haddad, and J. F. Peyre. Generative families of positive invariants in coloured nets sub-classes. *Lecture Notes in Computer Science; Advances in Petri Nets 1993*, 674:51–70, 1993.
7. J.M. Couvreur. The general computation of flows for coloured nets. In *proc of the 11th International Conference on Application and Theory of Petri-Nets*, Paris, June 1990.
8. J.M. Couvreur and S. Haddad. Towards a general and powerful computation of flows for parameterized coloured nets. In *9th European Workshop on Application and Theory of Petri Nets*, volume II, Venice (Italy), June 1988.
9. S. Evangelista. Helena, an efficient high level Petri nets analyser. Technical report, CEDRIC, CNAM, Paris, 2004.
10. S. Evangelista, S. Haddad, and J.F. Pradat-Peyre. New coloured reductions for software validation. In *Workshop on Discrete Event Systems*, 2004.
11. S. Evangelista, C. Kaiser, C. Pajault, J. F. Pradat-Peyre, and P. Rousseau. Dynamic tasks verification with QUASAR. In *Reliable Software Technologies - Ada-Europe 2005*, volume 3555 of *LNCS*. Springer-Verlag, 2005.
12. S. Evangelista, C. Kaiser, J. F. Pradat-Peyre, and P. Rousseau. Quasar: a new tool for analysing concurrent programs. In *Reliable Software Technologies - Ada-Europe 2003*, volume 2655 of *LNCS*. Springer-Verlag, 2003.
13. J. Ezpeleta, F. García-Vallés, and J. M. Colom. A class of well structured petri nets for flexible manufacturing systems. *Lecture Notes in Computer Science: 19th Int. Conf. on Application and Theory of Petri Nets, ICATPN'98, Lisbon, Portugal, June 1998*, 1420:64–83, June 1998.
14. F. Garcia-Valles and J. M. Colom. Implicit places in net systems. In *Proc. 8th Int. Workshop on Petri Net and Performance Models (PNPM'99), 8-10 October 1999, Zaragoza, Spain*, pages 104–113, 1999.
15. H. J. Genrich and K. Lautenbach. S-invariance in predicate/transition nets. In Pagnoni, A. and Rozenberg, G., editors, *Informatik-Fachberichte 66: Application and Theory of Petri Nets — Selected Papers from the Third European Workshop on Application and Theory of Petri Nets, Varenna, Italy, September 27–30, 1982*, pages 98–111. Springer-Verlag, 1983.
16. S. Haddad and C. Girault. Algebraic structure of flows of a regular coloured net. *Lecture Notes in Computer Science: Advances in Petri Nets 1987*, 266:73–88, 1987. NewsletterInfo: 27.
17. Serge Haddad and Jean-François Pradat-Peyre. New efficient petri nets reductions for parallel programs verification. *Parallel Processing Letters*, 16(1):101–116, 2006.
18. Lom Hillah, Fabrice Kordon, Laure Petrucci-Dauchy, and Nicolas Trèves. Pn standardisation: A survey. In *FORTE'06*, pages 307–322, 2006.
19. K. Jensen. Coloured Petri nets and the invariant method. In *T.C.S.*, volume 14, pages 317–336, 1981.
20. G. Memmi and J. Vautherin. Computation of flows for unary-predicates/transition nets. *Lecture Notes in Computer Science: Advances in Petri Nets 1984*, 188:455–467, 1985.
21. T. Murata. Petri nets : properties, analysis and applications. In *proceedings of the IEEE Vol 77*, number 4, pages 39–50, January 1989.

22. C. Pajault and J.F. Pradat-Peyre. Distributed colored petri net model-checking with cyclades. volume 4346 of *Lecture Notes in Computer Science*. Springer-Verlag, 2006. to appear.
23. W. Reisig. *EATCS-An Introduction to Petri Nets*. Springer-Verlag, 1983.
24. Wolfgang Reisig. Petri nets and algebraic specifications. *Theoretical Computer Science*, 80:1–34, 1991. NewsletterInfo: 38,39.
25. M. Silva Suarez, J. Martinez, P. Ladet, and H. Alla. Generalized inverses and the calculation of symbolic invariants for colored petri nets. *Technique et Science Informatiques*, 4(1):113–126, 1985. NewsletterInfo: 16,21,22.
26. J. Vautherin. Calculation of semi-flows for pr/T-systems. In *Int. Workshop on Petri Nets and Performance Models, Madison, Wisconsin*, pages 174–183, Washington, 1987. IEEE Computer Society Press. NewsletterInfo: 29.