Invariants and Home Spaces in Transition Systems and Petri Nets

Gerard MEMMI

LTCI, Telecom-Paris, Institut polytechnique de Paris

19 place Marguerite Perey F-91120 Palaiseau, France

Abstract

This lecture note focuses on comparing the notions of invariance and home spaces in Transition Systems and more particularly, in Petri Nets. We also describe how linear algebra relates to these basic notions in Computer Science, how it can be used for extracting invariant properties from a parallel system described by a Labeled Transition System in general and a Petri Net in particular. We endeavor to regroup a number of algebraic results dispersed throughout the Petri Nets literature with the addition of new results around the notions of semiflows and generating sets. Examples are given to illustrate how invariants can be handled to prove behavioral properties of a Petri Net. Some additional thoughts on invariants and home spaces will conclude this note.

Keywords: System modeling, design verification, Transition Systems, Petri Nets, invariant, reachability graph, temporal logic, linear algebra, semiflow, generating set, home space, liveness, fairness

Contents

1. Introduction

1.1. Motivations

Parallel programs, distributed systems, telecommunication networks, cyber-physical systems are complex entities to design, model, and verify. Even if they include analog components, this note will consider these entities as *Discrete Digital Systems* (DDS). Using formal verification at different stages of the system development life cycle is a strong motivation that spread throughout this note and provide us with rationale for many concepts, definitions, and behavioral properties. Discrete Digital Systems are often prototyped then simulated or emulated at great expense. Their behavior and outputs will be intensively tested and stressed over multiple kinds of inputs. Just to illustrate the magnitude of the importance of design verification, let us recall that integrated circuit design verification costs keep increasing and today, significantly surpass system design costs [1].

One problem is that DDS are often designed to operate during a long period of time or even indefinitely and continuously produce a stream of outputs. Another problem is that they often scale up by integrating components designed and developed independently. Last but not least of these problems, is

[1]https://blogs.sw.siemens.com/verificationhorizons/2021/01/06/part-8-the-2020-wilson-research-group-functional-verification-study/

that systems components are running concurrently, sharing common resources or objects, and generating an enormous amount of possible behaviors, many of them unforeseen during the design and integration phases. One then speaks of combinatorial explosion.

DDS description must cope with concurrency and complexity as well as scalability. One solution is to recourse to system modeling before prototyping and use abstraction mechanisms while being very careful to minimize loss of accuracy and to preserve wanted properties. For instance, designers may use genericity (via a kind of polymorphism or more simply parameterization) in their models at the cost of making their verification much more difficult and sometimes, almost intractable (see [Mem83], [BEI^{+}20], or section 7.2 for specific cases). System behavioral complexity depends upon the number of actions, the size, type, and structure of the data they can handle; however, this is the entanglement between concurrent or parallel actions that exacerbates their behavioral complexity and makes their design as well as their verification quite challenging.

Following R. Descartes' second and third principles [2], a system observer (be a designer, a developer, or a tester) will want to cope with this situation by dividing and regrouping properties into subsets [3] for which he will simplify and adapt the system model. Then, he will recompose the system step by step in a bottom up approach; detailing for each subset only relevant elements constituting what is sometimes called a system view or projection. Some of these behavioral properties will be called *invariants* the study of which is at the core of this note.

By the way, what is an invariant? We could paraphrase W. Bageholt [4] since

[2] "to divide each of the difficulties under examination into as many parts as possible, and as might be necessary for its adequate solution." then "by commencing with objects the simplest and easiest to know, I might ascend by little and little, and, as it were, step by step, to the knowledge of the more complex" in *Discours de la Méthode* (1637)

[3] these subsets designed for verification may differ from components ailing at an efficient architecture.

[4] "we know what it is when you do not ask us, but we cannot very quickly explain or define it." in *Physics and Politics* (1887) in:

on the one hand, so many papers use this notion without formal definition and on the other hand, so many papers introduce quite different definitions. How to best define and use invariants to analyze the behavior of a model? Can we not only prove that a formula is an invariant, but find a way in which they can be organized or concisely described, a way in which they can be discovered or computed? How can invariants be combined to represent a meaningful behavior? How can invariants be decomposed into simpler and verifiable properties? How to determine whether a given decomposition is more effective than another one with regard to formal verification?

In this note, we want to show how linear algebra or algebraic geometry can efficiently sustain invariant calculus as long as conceptual models can exhibit what we will call later in this note a topology allowing to deduce a system of equations with state variables as unknowns. In such a setting, linear algebra can also be applied and utilized to prove a large variety of behavioral properties.

One of our main motivation was to go beyond regrouping a number of algebraic results dispersed throughout the Petri Nets literature and to compare and better position these results in considering semirings such as $\mathbb{N}$ or $\mathbb{Q}^+$ then over a field such as $\mathbb{Q}$ especially regarding generating sets of semiflows ([Mem23]). In this regard, we were able to complete the current state of the art as accurately as possible with a range of new results on minimal semiflows, minimal supports, and finite minimal generating sets for a given family of semiflows.

At the same time, we try as much as possible to link models and conceptual models with actual system interpretations, behavioral properties, and motivations.

We sometimes looked at these conceptual models, basic properties, examples, and results from an historical perspective, which explains many of the quotes from the 1970s.

https://archive.org/ details/workslifeofwalte08bage/page/14/mode/2up

1.2. Outline

After providing several basic notations Section 1.3, we organize in Section 2 various informal definitions and concepts that will provide some intuition with what we mean by terms such as observer, model, or system under study. The notion of state variable and the fundamental notions of transitions and states together with the notions of trajectories, traces, and computations are introduced Section 3. This will support the definition and properties of home space described Section 3.4 and later of the notion of invariant Section 4.

The reader of this note is assumed to be somewhat familiar with Transition Systems described in Section 3.6, more especially with Petri Nets. Basic concepts and definitions are provided in Section 3.10 in order to produce a note as self-contained as possible. However, many more definitions, theoretical results, and examples can be found in [Kel76] or [FS01] for Transition Systems and in many books such as [Bra82] or [GV03] for Petri Nets.

Then, the notion of invariance is described in Section 4 trying to sort out various definitions found in the literature. Mutual exclusion is described and analyzed as a first example of invariant in Section 4.2. Then, this note focuses on how linear algebra relates to this basic notion in Computer Science, and how it can be used for extracting invariant properties from a parallel or concurrent system described by specific Transition Systems or Petri Nets.

In Section 5.2, semiflows, and the set $\mathcal{F}^+$ of all semiflows with non-negative coordinates of a Petri Net, the notions of generating sets, minimal semiflows, and minimal supports are then defined in Section 6. Moreover, in this section, it's important to relate semiflow theory to mathematical concepts from linear algebra, convex geometry, or discrete mathematics such as Sperner's theorem to provide with a bound for the number of minimal supports or Gordan's lemma about the existence of finite generating sets. This last connection has never before been attempted.

The three decomposition theorems of Section 6.4 have been first published in [Mem78]. Here, two of them are extended to better characterize minimal semiflows and generating sets over $\mathbb{N}$ and better covering $\mathbb{N}$, $\mathbb{Q}^+$, and $\mathbb{Q}$. They are at the core of this paper, since they have made it possible to consolidate results scattered in the literature in a meaningful way and are at the source of the new results presented in the following sections.

Succinct examples and counterexamples are provided to illustrate various results throughout Sections 6.4 to 6.6. Differences between minimal and canonical semiflows are made precise in the lemma 5 in Section 6.5, which is dedicated to this notion; the section ends with a theorem which states under which condition the number of canonical semiflows is infinite (Section 6.5.2).

The notion of minimal generating sets for semiflows described in Section 6.6 can already be found in [Mem77], then by Colom, Silva, and Teruel in [STC98] p. 319 and later by the same authors in [CST03], p. 68 or more recently, in [CMPAW09]. However, theorem 6 is new, to the best of our knowledge, and proves the coincidence between a least generating set and a minimal generating set which is not always true in order theory. Uniqueness of particular generating sets is presented in Section 6.6.3.

In Section 6.7, a table is summarizing main results, in particular showing differences [5] when considering $\mathbb{N}$, $\mathbb{Q}^+$, or $\mathbb{Q}$ and highlighting our contribution.

Examples are given to illustrate how invariants can be used to prove behavioral properties of a Petri Net in Section 7. A concrete example is drawn from the telecommunication industry Section 7.3; it illustrates how the choice of minimal semiflows of minimal support is an important factor in simplifying the proof of safeness and liveness for a given Petri Net.

Section 8 concludes and provides possible avenues of future research. At last, some additional thoughts will conclude this note.

1.3. Notations

The following basic notations are also introduced where they are used for the first time.

[5]A few inaccuracies found in the literature (e.g. in [CS89, CMPAW09]) stem from these differences.

Numbers and sets

$\mathbb{F}_2 = \{0,1\}$ is the 2-element field.

$\mathbb{N}$ is the set of natural numbers including 0, $\mathbb{N}_+$ is the set of natural numbers excluding 0.

$\mathbb{Z}$ is the set of integer, $\mathbb{Q}$ the set of rational, and $\mathbb{R}$ the set of real numbers. $\mathbb{Q}^+$ and $\mathbb{R}^+$ are the semirings associated with $\mathbb{Q}$ and $\mathbb{R}$ respectively i.e. the sets of non-negative rationals and reals.

If E is a set, $|E|$ denotes the cardinal number of E.

ω denotes infinity and is absorbing for the addition ($\omega = \omega{+}1$). We abusively use the same symbol '0' to denote $(0, ..., 0)^\top$ of $\mathbb{N}^n$, $\forall n \in \mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$.

For any countable set $\mathbb{X}$, resulting of an n-ary Cartesian product, $\mathbb{X}_\omega = \mathbb{X} \cup \{(\omega,\ \omega, ...,\ \omega)\}$ where ω is repeated n times, which is quite different from $\mathbb{X}^\omega = \mathbb{X} \times \mathbb{X}...\ \omega$ times. With these conventions, it becomes clear that if A and B are two sets then $(A \times B)_\omega \neq A_\omega \times B_\omega$.

Relations, functions, and vectors

If E and F are two sets, and a *relation* r from E to F is a subset of the Cartesian product $E \times F$. Several notations can be used to express that x of E is in relation with y of F by r: we can use the infix notation xry or $x \xrightarrow{r} y$, but also say that the pair (x, y) belongs to r and define $r(x) = \{y \in F | (x, y) \in r\}$ as the *image* of x under r.

The domain of r is defined as: $\mathtt{Dom}(r) = \{x \in E | r(x) \neq \varnothing\}$ and the image of r (sometimes called codomain) is defined as: $\mathtt{Im}(r) = \{y \in F | \exists x \in E,\ (x, y) \in r\}$.

Given a relation f from $E \times F$ to G and $x \in E$, $f(x, \cdot)$ denotes the relation from F to G such that: $\forall y \in F, f(x, \cdot)(y) = f(x, y)$. Similarly, given $y \in E$, $f(\cdot, y)$ denotes the relation from E to G such that: $\forall x \in F, f(\cdot, y)(x) = f(x, y)$; and can be seen as a projection of f on E.

A *partial function* f from E to F is a relation such that $\forall x \in E,\ |f(x)| \leq 1$, then f is a *function* (sometimes also called *mapping*) when it is a partial function and $\mathtt{Dom}(r) = E$.

If f is a partial function from E to $\mathbb{N}$ or $\mathbb{F}_2$, then the *support of* f is denoted by $\|f\|$ and is defined by $\|f\| = \{x \in \mathtt{Dom}(f) \mid f(x) \neq 0\}$.

With x_i denoting the i^{th} coordinate of a vector x, two vectors f and g can be compared and $f \leq g$ if and only if $f_i \leq g_i$ for all coordinate of the two vectors. f^T denotes the transpose of f and $f^T g$ denotes the scalar product of f and g, we have $f^T g = \sum_i f_i g_i$.

Words

Let T be an alphabet, T^* denotes the free monoid generated by T that is to say the set of finite words over T. If T is ordered such that $T = \{t_1, ..., t_m\}$ and r is a word over T^*, $|r|$ is its length and $\overrightarrow{r}$ (sometimes called the Parikh vector of r) is such that $\overrightarrow{r_i}$ is the number of occurrences of the letter t_i in the word r and $\overrightarrow{r} \in \mathbb{N}^{|T|}$. A word has a notion of support similar to that of function: $\|r\| = \|\overrightarrow{r}\| = \{t_i \in T \parallel \overrightarrow{r_i} > 0\}$

T^ω is the set of infinite words and $|r| = \omega$ if and only if $r \in T^\omega$; and $T^\infty = T^* \cup T^\omega$ is the set of finite and infinite words.

Various abbreviations

M stands for model, CM for conceptual model, TS for Transition System, LTS for Labeled Transition System, PN for Petri Net, VAS for Vector Addition System. $\langle M, q_0\rangle$ denotes a model M paired with an initial state q_0.

$\mathcal{SV}$ is denoting the set of *state variables* and $|\mathcal{SV}| = d$ denoting the number of state variables. $\mathcal{SV}$ can be ordered and we have $\mathcal{SV} = \{x_1, ..., x_d\}$.

T is denoting the set of named or labeled transitions and $|T| = m$.

Q(M), $RS(M, q_0)$, $RG(M, q_0)$, $LRG(M, q_0)$ denote the *state space*, the set of *reachable states* (i.e. Reachability Set), the Reachability Graph, the Labeled Reachability Graph of $\langle M, q_0\rangle$ respectively. These sets will more simply be denoted Q, RS, RG, and LRG respectively when there is no ambiguity.

2. Observer, systems under study, models, and conceptual models

We cannot address the notions of invariant and home space without first describing even succinctly the basic objects and concepts motivating our study. A model can support in various different ways the description and the analysis of a program or of all kind of systems. In our case, a model must describe its *behavior* that is to say how a DDS evolves over time from one state to the next one as it is running and performing various operations or actions.

2.1. Observer

The notion of model cannot in our opinion, be separated from the notion of observer. An observer may have different additional roles which can influence the kind of information he needs to observe; he can be a designer, an architect, a developer, a test or a maintenance engineer. An *observer* is interacting with the model before and after its progression during which he is only observing the model evolution. His interaction consists in selecting the level of information he can or wants to extract from the model. To do so, he defines a model environment, instrumenting the model and defining output data, preparing input data, tuning parameters, selecting an observer clock frequency (i.e. the level of granularity or accuracy with which he will be able to extract or sample information). In essence, the observer wants to draw historical information about the dynamic functioning of the model:

- at what time, under which circumstances does such element change the value of such variable?
- Is the observed evolution of the model in accordance with the observer's expectations expressed under the form of behavioral properties or in accordance with the behavior of the actual system under study?

As a designer or an architect, the observer may want to proceed by reification with a new version of the model, by refining the model environment with more inputs, or by increasing of his clock frequency and be able to observe the model with more details. He will be satisfied if the two levels of observation are somewhat compliant; and he will look for an improved degree of determinism, in other words that the past and present has some predictive value [6].

[6]This is not without reminding the stoic doctrine that Seneca expresses in [Sen10] book2, 32, 4: "Everything that happens is a sign of some future event. Chance events and purposeless, chaotic ones do not admit of divination; where there is order, there is also predictive force."

As a maintenance engineer he may want to align what he observed on the model with the behavior of the actual system under study .

The notions of system under study, model, and conceptual model are now informally introduced. Models and conceptual models will be introduced later in Section 3 with formal definitions together with a few behavioral properties.

2.2. Systems under study (SuS)

We will call *system under study* the original entity be a program or a digital system to be described then verified. We are interested in systems evolving or progressing most of the time indefinitely (as in UNITY [CM88]). They will be performing statements, actions, activities, or reacting to events. They will be handling resources which are consumed, tested, modified, shared, or produced in discrete quantities while statements, actions or activities are being performed. Some of these actions could be running in parallel when they are independent of each other or in concurrency when they race for a shared resource.

In this note, we will consider that a system under study is able to perform a finite number of different elementary or *indivisible* actions (i.e. actions that are not a composition of observable actions) over a finite set of different types of resources. This does not prevent systems to present infinite behaviors or being able to produce a resource of a given type in an infinite quantity. The dynamic progression of an SuS is modeled by a notion of computation i.e. a sequence finite or infinite of actions altering a set of resources.

At last, we will not consider the notion of time in terms of dates or duration. Therefore, in this note, the words *action* (which often bears a meaning of duration) or *event* (which often is understood as instantaneous) will be synonymous (see for instance [GK18], for an illustration of the difference between the two notions where time is taken into account).

2.3. Models (M)

We will call *model* the formal description which will support the design, the analysis, or the verification of the behavior of the system under study by an *observer.* This description is discrete and must support the representation of the dynamic evolution or progression of the system under study.

Model and observer are linked together. The model is defined by the observer in accordance to the level of observation he wants or can reach. Of course, a model is not *fully* equivalent to the system under study it describes. A model is developed with the intent of describing and verifying a specific set of properties not any property the system under study may or must satisfy. A "good" model will retain just enough details and will be as small as possible in order to conduct as an effective verification as possible and at the same time, it must be large enough to convincingly behave like the system under study in regard to the specific set of properties to be proven. It must be recalled here, that most of the time, a verification algorithm complexity depends among other parameters on the size of the model as wall as the level of interactions between its different operations. To this regard, one can speak of an *art of system modeling* especially when considering techniques such as projections, reductions (see [Bra82, CST03]), or rewriting (see [DJ90]) that can be used to simplify the model while preserving its desired properties. However, describing in details these very interesting techniques are out of the scope of this note.

2.4. Conceptual models (CM)

Like programs are written in a programming language, models are described in a *conceptual model.* Sometimes the terms abstract machine (as in CAML [7]), computational model, or formal model are used instead ([FS01] cites the two later terms indifferently, for instance). With this in mind, Turing machines are a conceptual model as well as automata or Kripke structures [Kri63]. A specific automaton which would represent the addition of two bounded integers would be a model of an adder and we use to say that such a model is an automaton in a kind of metonymy where we use the same name for a specific model and the underlying conceptual model.

[7]https://caml.inria.fr/about/history.fr.html

In this note, we will be interested in CMs expressive enough to describe the dynamic progression (also called behavior or evolution) of a system under study. In the digital world we are interested in, this progression is discrete and characterized by a notion of state taking its values in countable sets. CMs will include a notion of transition (or action) that will describe how the model progresses from one state to the next.

3. States and Transitions

3.1. State variables, states, and state spaces

3.1.1. State variables

A state variable can represent any kind of information even outside of the system itself. In some of his own examples, Hoare [Hoa15] includes what he calls the interaction with the system environment to take such elements into account. For instance, an ambient temperature, an altitude, or a number of users can qualify as state variables despite the fact that they very likely are outside the system under study, but interact with it to the point where they can change its outputs or behavior.

A state variable can be a simple Boolean variable or a complex data structure controlling all sort of synchronization mechanism: queues, shared resources, clocks, watchdogs,...

$\mathcal{SV}$ will denote the set of state variables of a given model and unless specified otherwise, $\mathcal{SV}$ will be finite with $|\mathcal{SV}| = d$. For convenience, we may want to order $\mathcal{SV}$ such that $\mathcal{SV} = \{x_1, ..., x_d\}$. A state variable x_i has a type that defines the range of values D_i it can take. State variable values may be in $\mathbb{R}$ as in[CM88] or [TRSS01]; however in this note, most of the time, we will limit ourselves to values in countable sets particularly $\mathbb{N}$, $\mathbb{F}_2$, or any finite enumerated set.

The observer is central in the selection of state variables: as long these variables are useful and accessible (directly or indirectly by means of instrumentation in the SuS), or can influence the progression of the system and alter the properties he wants to analyze. Concerning DDS, authors often distinguish two kind of state variables:

• the memory, the registers, data variables witnessing (such as outputs) or influencing (such as inputs) the system progression, or data variables which can support monitoring resource variation pertinent to properties to be proven;

• the control variables such as Boolean variables to determine a possible choice between two pieces of code, semaphore variables to control access to shared resources, labels, or ordinal counters to point towards the next possible statements that can be executed by the system under study.

We say that $\mathcal{SV}$ is *complete* if and only if the model progression depends uniquely on the values of the current state. Just as an example, a model can progress according to its past history rather than its current state. In this regard, it can be pointed out that historical data (i.e. model progression) can be encoded in a specific state variable to complete $\mathcal{SV}$.

3.1.2. States and state spaces

Most conceptual model designed to analyze system behaviors and address concurrency will manipulate explicitly or implicitly a notion of *state*. A state must contain enough information to monitor or even predict the system progression at a level of observation sufficient to analyze and determine whether a set of desired properties is met or not. Briefly, a state is defined by a set of *state variables* determined by the observer .

In [Ash75] or in [Kel76] and in many other theoretical computer science papers, a state will be structured as a couple of vectors, separating the two kind of state variables we have just mentioned (i.e. observation or memory and control).

A *state* q is a mapping from $\mathcal{SV}$ to the set of potential values taken by the elements of $\mathcal{SV}$. This set is called the *state space* and is denoted by Q when there is no ambiguity, and *Q(M)* when it is necessary to relate it to a particular model *M*. In this note, we will assume $Q(M)$ countable. More precisely, if $x_i \in \mathcal{SV}$ varies in the set D_i then $q(x_i) \in Q$ where $Q = \prod_{i=1}^{i=d} D_i$. A *state* q is seen as an element of Q. Once $\mathcal{SV}$ is ordered, it can also be seen as a vector of dimension d. The most convenient notation between q_i, $q(x_i)$, or x_i will be adopted in compliance with the notation of the model description.

Regarding a system under study, it is often sufficient to consider amounts of d different *state variables* (including Boolean variables) modeling the number of resources or the truth values of control elements the system under study disposes of. Henceforth, we may assume $Q = \mathbb{N}^d$ by default and a state q can be then seen as a vector of $\mathbb{N}^d$ unless more information is made available. For instance, if we know that there are d_1 boolean state variables and d_2 integer state variables such that $d = d_1 + d_2$, we have $Q = {\mathbb{F}_2}^{d_1} \times \mathbb{N}^{d_2}$.

At last, two subsets of Q are usually distinguished: $Init$ denoting the set of *initial states* i.e. the states from which the model begins its progression, $Term$ denoting the set of *terminal states* i.e. the states where the model progression halts (or at least, where the observer stops his observation). In the next sections, models will often be paired with a single initial state q_0 with $Init = \{q_0\}$; and unless specified otherwise, we will have: $Term = \varnothing$ since we are interested by models with infinite behaviors.

3.2. Transitions, traces, and languages

A transition is a key element for any conceptual model. It is modeling an indivisible event, statement, or action of the SuS. A *transition* t is a relation in $Q \times Q$. If $(q, q') \in t$, then we say that q' is directly reachable from q. We then write $q \xrightarrow{t} q'$. If $q \in \texttt{Dom}(t)$, then we write $q \xrightarrow{t}$. When the name or label of the transition is unknown, we write $q \to q'$ and $q \to$ respectively.

Most of the time, transitions are given a name or a label, the set T of transition labels or names is then finite and by convention $|T| = m$.

A *sequence of transitions* is a word of $T^* \cup T^\omega = T^\infty$. In a given model, only a subset of T^∞ can represent actual computations. This is why a notion of trace is introduced. A *trace* σ is a relation in $Q \times Q$ which is a composition of transitions say $t_1,\ t_2, ...t_n$ such that $(q, q') \in \sigma$ if and only if $\exists\ q_1, q_2, ...q_{n+1} \in Q$ such that $\forall i \in \{1, ...n\},\ (q_i \xrightarrow{t_i} q_{i+1})$ with $q_1 = q$ and $q_{n+1} = q'$. We write

$\sigma = t_0 t_1 ... t_n$ as a word of T^∞. We sometimes say that σ begins at q and ends at q'. We then write $q \xrightarrow{\sigma} q'$ and $q \xrightarrow{\sigma}$ if and only if $q \in \mathtt{Dom}(\sigma)$. If (q, q') belongs to an unknown sequence of transitions, we write $q \xrightarrow{*} q'$ and $q \xrightarrow{*}$ respectively, $\xrightarrow{*}$ denoting the reflexive transitive closure of $\rightarrow$.

A sequence of transitions may or may not be a trace in the model under consideration; it may depend on the chosen initial state or it may just be impossible to compose the transitions in the order specified by the sequence of transitions.

Given a model M and a subset $Init$ of states, $\mathcal{L}(M, Init)$ is the set of all possible traces beginning at $q \in Init$ and denotes the *language* of $\langle M, Init\rangle$. As long as $Term = \varnothing$, a language $\mathcal{L}(M, Init)$ is closed for the prefix relation *pre*: $\forall w \in \mathcal{L}(M, Init), pre(w) = \{v \mid \exists u \in T^\infty, w = vu\} \subseteq \mathcal{L}(M, Init)$. This property is valid for Petri Nets [Bra82].

3.3. Reachability, trajectory, and structure

3.3.1. Reachability sets

As we learn how a model progresses over time, a relation of reachability can be introduced between states and we will say that a state b can be *reached* from a state a (sometimes, b is said *accessible* from a as in automata theory [Sak09] or [Eil74]) if there a mean for the model to progress from a to b and we write $a \xrightarrow{*} b$ to follow the notation introduced in Section 3.2. The set of all reachable states from an initial state q_0 is traditionally called the *reachability set* and is denoted by RS when there is no ambiguity and $RS(M, q_0)$ when we consider the specific model M paired with the initial state q_0. This definition immediately raises the question of knowing which state is in $RS(M, q_0)$ which state is not. At the level of conceptual models, this question is better known as the reachability problem:

the reachability problem for a conceptual model CM is decidable if there exists an algorithm taking as inputs a model described in CM, a state a and a state b, able to determine whether or not b is accessible from a.

Of course, for a given model *M*, we have $\forall q \in$ *Q(M)*, *RS(M, q)* $\subseteq$ *Q(M)* and $Q(M) = \bigcup_{q \in Q(M)} RS(M, q)$.

If $Init$ is a subset of Q, $RS(M, Init)$ will denote $\bigcup_{q \in Init} RS(M, q)$.

Similarly, we can define the subset of states able to reach a given state q: $RS^{-1}(M, q) = \left\{ q' \in Q \mid q' \overset{*}{\rightarrow} q \right\}$ and extend the definition to a subset A of Q: $RS^{-1}(M, A) = \bigcup_{q \in A} RS^{-1}(M, q)$. A state q is *isolated* if and only if $RS(M, q) = RS^{-1}(M, q) = \{q\}$ and $Isol$ denotes the set of isolated states.

Property 1. Let's Q be the state space of the model M with $\mathcal{SV}$ its set of state variables x_i each varying in the domain D_i, we have:

$Q = \prod_{i=1}^{i=d} D_i = \bigcup_{q \in Q} RS(M, q) = \bigcup_{q \in Q} RS^{-1}(M, q) = \bigcup_{t \in T} \mathtt{Dom}(t) \cup \bigcup_{t \in T} \mathtt{Im}(t) \cup Isol.$

3.3.2. Reachability graph

To explore and analyze the reachability set in a concise manner and support the resolution of the reachability problem, a *reachability graph* $RG(M, q_0)$ (RG when there is no ambiguity) can be constructed where the set of vertices is in bijection with $RS(M, q_0)$ that is to say that each vertex n is labeled by exactly one element q of $RS(M, q_0)$ and that any reachable state q is the label of exactly one vertex. In the next sections, we will not make distinction between a vertex and its label. There will be an edge in $RG(M, q_0)$ from vertex a to vertex b if state b is directly reachable from state a in $\langle M, q_0 \rangle$ with no intermediary state (sometimes it is said with no intermediary observable state) when going from a to b, and we write $a \rightarrow b$ to follow the notation introduced in Section 3.2. This kind of progression will then be said *indivisible* (to the eyes of the observer) in the corresponding system under study. $RG(M, q_0)$ can be a multi-graph when there exists several different direct progression from a to b [8].

[8]Actually, the observer could be able to observe multiple different progressions from a to b. That would define a subgraph of RG or more precisely, a sub-model with a as its initial state and b as its terminal state containing strictly all paths from a to b. This could be seen as a kind of weak non-determinism since the system would have the choice to pick one path or another to progress from a to b.

A directed path will join a state a to a state b if and only if b is reachable from a. This directed path can be seen as a sequence of consecutive states which is called a *trajectory*.

A *root* is a vertex of RG with no predecessor. Similarly, a *sink* is a vertex with no successor. If it is clear that a root can only be associated with an initial state, it is worth to point out that first, an *initial state* is not necessarily a root; second, a sink is not necessarily associated with a terminal state and can be interpreted as an unexpected system halt (sometimes called a deadlock); third, a terminal state is not necessarily a sink.

The reachability graph is a directed multigraph with one source strongly component containing the vertex labeled by q_0 as long as we consider models paired with q_0 a unique initial state.

In order to more precisely figure out how to move from one state to another, the notions of transition and Transition System will be introduced in Section 3.6 where in addition, the definition of reachability graph will be refined further.

As soon as a conceptual model allows organizing states we will say that it possesses a *structure*, notion that we will keep informal and which is referring to the information detained about states. Therefore, a reachability graph is an example of a structure. A structure can hold more information about the states of systems under study and is key to support efficient behavioral analysis. In particular, the reachability graph is a powerful tool to study all kind of properties such as reachability questions or to solve the reachability problem. Trajectories can be infinite even when RS is finite. The reachability graph also can be infinite for two reasons: there is an infinite multiplicity of edges between two successive vertices or there is an infinite number of vertices. In this note, we will limit ourselves in considering reachability graphs where vertices have a finite degree (called *locally finite* in [Die17] [9]). We may have an infinity of vertices and this is precisely why the reachability problem is such an interesting decidability problem. The interested reader can look at [BFM18] for reachability

[9]or sometimes *finitely branching* as in [FS01]

graphs where vertices may have an infinite degree.

3.4. Home spaces and home states

The notion of home space was first defined in [Mem83] for Petri Nets. Here, we extend its definition without difficulties as soon as a conceptual model allows building up the reachability set and its associated reachability graph.

Given a model M its associated state space Q and a subset A of Q, we say that a set *HS* is an *A-home space* if and only if for any progression of the model M from any element of A, there exists a way of prolonging this progression and reach an element of *HS*. In other words:

A set *HS* is an *A-home space* if and only if $\forall q \in RS(M, A)$ $\exists h \in HS$ such that h is reachable from q (i.e. such that $q \xrightarrow{*} h$). In [JL22], we can find for Petri Nets an equivalent definition : *HS* is an *A-home space* if and only if $RS(M, A) \subseteq RS^{-1}(M, HS \cap Q)$.

We immediately point out that we must have $HS \cap RS(M, A) \neq \varnothing$, however, HS is not necessarily included in Q. If $RS(M, A) = Q$ then we say that HS is a *structural home space.* It means that $\forall Init \subseteq Q$, HS is a *Init*-home space. If $\exists q \in Q$ such that $A = RS(M, q)$, we say that *HS* is a *home space* of (M, q).

When *HS* is reduced to one single state $\{s\}$, we say that s is an *A-home state.* This last definition is expanded from the definition given in [Bra82] p.59 or in [GV03] p. 63 for Petri Nets. In many systems, the initial state q_0 is defined as an *idle* state from which the various capabilities of the system can be initiated. This important property is usually guarantied by a reset function which can be modeled in a simplistic way by a transition r such that $\forall q \in Q,\ q_0 \in r(q)$. However, adding too much complexity to RG (one edge per node), this function is most of the time abstracted away.

Given a subset of states A, it is not easy to prove that a given set is an A-home space. This question is addressed in [JL22] and is proven decidable for Petri Nets for home state but is still open in the general case. In this regard, it may be worth noticing the straightforward following properties:

Property 2. any set of states containing a home space (or a home state for the matter) is also a home space. Therefore, the union of two home spaces is a home space.

However, the intersection of two home spaces is not necessarily a home space (save when the two home spaces contain RS).

Property 3. if there exists a sink $q_f \in A$ then it belongs to any A-home space.

For the two following properties, we consider a model M paired with an initial state q_0 and $A = RS$.

Property 4. any home space has at least one element in each sink strongly connected component of the reachability graph [10].

Property 5. The three following statements are equivalent:

(i) the initial state is a home state (then there is no sink),

(ii) any reachable state is a home state,

(iii) the reachability graph is strongly connected.

3.5. Boundedness

As soon as $\mathcal{SV}$ and its associated state space are defined, the observer of the corresponding model M will want to understand and if possible, simplify and reduce the state space Q in which state variables actually vary. Furthermore, he will want to see if their progression follow a simple rules or equations, or if there exists an upper bound for a subset of $\mathcal{SV}$ during the potentially infinite progression of M.

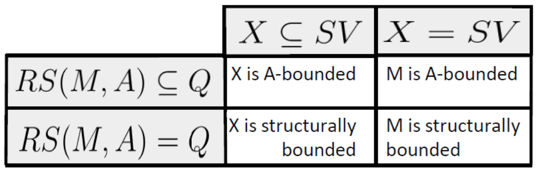
We consider $X \subseteq \mathcal{SV},\ A \subseteq Q$, we will say that X is *A-bounded* (or bounded when there is no ambiguity) if and only if:

$\exists k \in \mathbb{N}$ such that $\forall x \in X,\ \forall q \in RS(M, A),\ q(x) \leq k$.

[10] It is easy to prove that this property holds even when the reachability graph is infinite considering that the definitions of sources, sinks, or strongly connected components are the same as in the case where the directed reachability graph is finite.

If $X = \mathcal{SV}$ then we say that M is A-bounded. If $RS(M, A) = Q$ then X is *structurally bounded*. If $X = \mathcal{SV}$ *and* $RS(M, A) = Q$ then M is structurally bounded.

These somewhat overlapping definitions are positioned in relation to each other in Figure 1.



	$X \subseteq SV$	$X = SV$
$RS(M,A) \subseteq Q$	X is A-bounded	M is A-bounded
$RS(M,A) = Q$	X is structurally bounded	M is structurally bounded

Figure 1: Different definitions of boundedness according to the fact that every state variable is bounded or not for every possible state or not.

The fact that a state variable variation follows a rule or an equation along the model progression will be illustrated in examples in Section 7. Structural boundedness will be discussed again in particular in Section 5.3.2.

3.6. Transition Systems (TS) and Labeled Transition Systems (LTS)

Many papers use the notion of Transition Systems (sometimes State Transition Systems as in [CM88]) without even providing a formal definition. Hereunder, we use the definitions introduced in [Kel72, Kel76], then definitions by [SSM03] and [FS01] will be introduced. We point out that the definition of Transition Systems given in [Lie76a] is in fact equivalent to the definition that will be provided for Petri Nets in Section 3.10.1.

A *Transition System* is a pair $TS = \langle Q, \rightarrow \rangle$ where: Q is a non empty *state space*, $\rightarrow \subseteq Q \times Q$ is a binary relation on Q and is called the *set of transitions*. $(q, q') \in \rightarrow$ is denoted by $q \rightarrow q'$.

Q as well as $\rightarrow$ can be countably infinite. Neither states nor transitions are precisely defined. No wonder the reachability problem is not decidable in general. However, a large class of Transition Systems called Well-Structured Transition Systems (WSTS) were defined in Section 3.8 for which questions close to the reachability problem are decidable thanks to a particular ordering over Q which considerably enrich the conceptual model's structure.

At this stage, Transition Systems are a conceptual model with no topology. States are just elements of Q with no details (no information is provided on state variables in the definition of a Transition System). A model is simply described by successive states it can take during its progression (i.e. traces). Transitions have no names to help reporting about possible properties. Moreover, we want to handle a finite set of identifiable transitions since our systems under study are able to perform a finite set of possible indivisible actions. To this regard, we consider the notion of Labeled or named Transition System (*LTS*) that we borrow from [Kel76] but that we can find in many other publications such as [JSS20].

A *Labeled (or Named) Transition System* is a triple $LTS = \langle Q, T, \rightarrow \rangle$ where: Q is a non empty set of states, T is a non empty set of labels and $\rightarrow \subseteq Q \times T \times Q$ relates labels with states.

As for Transition Systems, Q can be infinite as well as T. The reader will have recognized the key elements of the definition of an automaton (as defined p.51 of [Sak09]) where T is an alphabet and where only initial and terminal states are missing to the definition. Actually, one element q_0 of Q is often distinguished and called *initial state*, however, terminal states are scarcely used since most of the time, the system under study is supposed to run forever.

Henceforth, we will call *transition* a label t of T; t will also denote a relation in $Q \times Q$ such that $(q, q') \in t$ if and only if $(q, t, q') \in \rightarrow$, in that case, we will use the notations $q \xrightarrow{t} q'$ or $q' \in t(q)$. We can compose transitions to form computations (see [Sak09] p.54) and sequences of transitions : if $(q_{i-1}, q_i) \in t_i$ for $i \in \{1, ..., n\}$ we then form $(q_0, t_1, q_1), (q_1, t_2 q_2), ...(q_{n-1}, t_n, q_n)$ defined as a *computation* (or run) and denoted by: $q_0 \xrightarrow{t_1} q_1 \xrightarrow{t_2} q_3 ... \xrightarrow{t_n} q_n$. We then say that $\sigma = t_1 t_2 ... t_n$ is a trace as defined in Section 3.2 and $(q_0, q_1, ...q_n)$ is a trajectory as defined in Section 3.3. We write $q_0 \xrightarrow{t_0 t_1 ... t_n} q_n$ or $q_0 \xrightarrow{\sigma} q_n$, or sometimes

$q_n \in \sigma(q_0)$. We also say that q_n is reachable from q_0 [11].

We must keep in mind that a transition is a relation and in general, cannot be seen as a function or even a partial function from Q to Q (we will see however, that Petri Nets assume that transitions are partial function).

3.7. Labeled Reachability Graph (LRG) and behavioral properties

Now that we have names for identifying transitions, we can move to the notion of Labeled Reachability Graph from the Reachability Graph defined in Section 3.3. Moreover, we can speak of behavioral properties focusing on frequencies or patterns of sequences of transitions such as liveness.

3.7.1. Labeled Reachability graph (LRG)

Given a Labeled Transition System $M = \langle Q, T, \rightarrow \rangle$ paired with an initial state q_0, $LRG(M, q_0)$ denotes the reachability graph of $\langle M, q_0 \rangle$. $LRG(M, q_0)$ is a directed connected multigraph $(RS(M, q_0), E)$ where $RS(M, q_0)$ is the set of vertices and E the set of edges such that $(x, y) \in E$ if and only if $\exists t \in T$ such that $x \xrightarrow{t} y$. The edge (x,y) is then labeled by t and we then say that t *appears as a label* in $LRG(M, q_0)$.

$LRG(M, q_0)$ differs from $RG(M, q_0)$ only by the addition of labels on its edges. $LRG(M, q_0)$ (LRG for short) can be an infinite graph with an infinite number of ends even if T is finite and LRG locally finite (see [Die17] chapter 8 for a comprehensive study on infinite graphs). Of course, there is a path from q_0 to any vertex x of LRG (which makes LRG a connected directed graph with at most one root).

The set of *computations* of $\langle M, q_0 \rangle$ is exactly the set of labeled paths of LRG. Let $c = x, q_1, ...q_n, y$ be such a path from x to y and $t_0, t_1, ..., t_n$ be the sequence of labels on the edges of this path. The *sequence of states* $x, q_1, ...q_n, y$ is said to

[11] In [FS01], a computation is a maximal sequence of transitions. This extends the definition in [Sak09] where a successful computation starts from the set of initial states to reach a state in the set of terminal states.

be a *trajectory* while the *sequence of transitions* $t_0, t_1, ..., t_n$ constitutes a *trace* also denoted by the word $t_0 t_1 ... t_n$ of T^*.

3.7.2. Determinism as a relation between traces and trajectories

A trace $t_1 t_2 ... t_n$ *corresponds* to a trajectory $a_0, a_1, ... a_n$ or reversely, a trajectory $a_0, a_1, ... a_n$ corresponds to a trace $t_1 t_2 ... t_n$ if and only if $\exists\ q \in Q$ such that we have the labeled path $a_0 \xrightarrow{t_1} a_1 \xrightarrow{t_2} a_3 ... \xrightarrow{t_n} a_n$ in $LRG(M, q)$.

Let us point out that traces and trajectories are not in general in a one-to-one correspondence. Several different trajectories may correspond to the same trace even if they start from the same state and several different traces may correspond the same trajectory. For a given model, a one-to-one correspondence would at least involve the two following properties:

Definition 1. We say that an LTS is *deterministic* if and only if the two following properties are satisfied:

P1 For any given sequence of states ρ there exists at most one sequence of transitions to form a computation with ρ.

P2 for any given state q, a given sequence of transitions σ there exists at most one sequence of states starting at q to form a computation with σ.

To obtain such a one-to-one correspondence would require to put some restrictions on the set of transitions such as the ones of the following lemma which makes the properties P1 and P2 easily verifiable.

Lemma 1. *In an LTS, we have the two following properties:*

L1 Any sequence of states has at most one corresponding trace if and only if: $\forall t_1, t_2 \in T,\ \forall q \in Q,\ t_1(q) \cap t_2(q) = \varnothing$

L2 Any sequence of transitions has at most one corresponding trajectory starting at a given state q if and only if any transition is a partial function.

The proofs are somewhat similar for L1 and L2, so they are jointly performed. If we consider a sequence of only two states q_i and q_f, according to P1, there is at most one trace to form a unique computation. This trace can only be a single transition. So, there is at most one transition between any couple of states therefore $\forall t_1, t_2 \in T,\ \forall q \in Q,\ t_1(q) \cap t_2(q) = \varnothing$. Similarly, if we consider a sequence of one transition t and one state q_i, from P2 we can form at most one computation, therefore from q_i t can reach at most one state. Hence, t is a partial function.

Reversely, we consider a sequence of states $\rho = a_0, a_1, ...a_k$, two traces $\sigma = t_1, ...t_k$ and $\sigma' = t'_1, ...t'_k$ forming a computation with ρ, and i the first index such that for $j < i, t_j = t'_j$ and $t_i \neq t'_i$, then we would have a contradiction with $t_i(q_i) \cap t'_i(q_i) = \varnothing$.

At last, we consider a sequence of transitions $\sigma = t_1...t_k$ and two trajectories $\rho = q_i, ...q_k$ and $\rho' = q_i, q'_1, ...q'_k$ starting at the same state q_i and forming a computation with σ and i the first index such that for $j < i, q_j = q'_j$ and $q_i \neq q'_i$, then we have $\{q_i, q'_i\} \subseteq t_i(q_{i-1})$ which would be a contradiction with the fact that t_i is a partial function. $\square$

It is worth pointing out that the property P2 means in particular that given an initial state q_i and a sequence of transitions σ there is at most one state q_f such that $q_i \xrightarrow{t} q_f$. This is consistent with the intuitive notion of determinism [12]. In [Eil74], deterministic machines are defined with conditions similar to the two properties L1 and L2 of lemma 1. We also found in [Sak09] p.77 a property slightly different from L2 of lemma 1 stating that any deterministic automata (briefly meaning that any transition is a partial function) is an unambiguous automata (briefly meaning that any sequence of transitions starting at an initial state and finishing at a terminal state forms at most one computation).

Determinism must be considered a good property especially when it is about to make sure that a test is reproducible. However, many authors define non-determinism in their conceptual model to take into account statements that are independent and can run in parallel [CM88] or to represent a possible ar-

[12]For instance in [CM88], the 'N' in UNITY stands for non-determinism with the following: "Different runs of the same program may execute statements in different orders, consume different amounts of resources, and even produce different results."

bitrary choice (between guarded commands [Dij75] or between sequential processes [Hoa15]).

3.8. Well Structured Transition Systems (WSTS)

Motivated by extending the domain of CMs for which boundedness and other behavioral properties would be decidable, Finkel (see [FS01] in particular) introduced the conceptual model of well structured transition systems as a subclass of TS or LTS. To this regard, the author enriched a TS or an LTS with an order relation denoted $\leq$ in $Q \times Q$. Here we give the definition with an LTS notation:

A labeled transition system is a *well structured transition system* (WSTS) $M = \langle Q, T, \rightarrow, \leq \rangle$ where $\langle Q, T, \rightarrow \rangle$ is an LTS and:

$\leq$ is reflexive, transitive, and such that for any infinite sequence of states $\rho = q_1, ...q_k, ...$ there exist two indices i, j such that $i < j$ and $q_i \leq q_j$. We then say that $\leq$ is a *well-quasi-ordering* (wqo),

$\forall (q, q', s) \in Q^3$ such that $q \leq q'$ and $t \in T$ such that $s \in t(q)$ then there exist a sequence $\sigma \in T^*$ and a state s' such that $s' \in \sigma(q')$ and $s \leq s'$. We then say that $\leq$ is *upward compatible* with $\rightarrow$.

The definition holds with using a TS instead of an LTS (i.e. with no name for the set of transitions).

3.9. Behavioral properties

In this section, we consider a pair $\langle LTS, q_0 \rangle$ where q_0 is the initial state of an LTS, a labeled transition system equipped with $\leq$ an ordering relation in Q.

3.9.1. Monotonicity and repetitivity

A first property that we can already find in [Bra82] p. 43 or in [Val81] relatively to PNs is the property of monotonicity. It was expanded in [Fin20, FS01] for LTS with the notion of strong monotonicity.

We say that a transition t is *strongly monotone* if and only if $\forall (q, q') \in Q^2$ such that $q \leq q'$ and $\texttt{Dom}(t)$, we have $q' \in \texttt{Dom}(t)$ and $\forall p \in t(q) \exists p' \in t(q')$ such that $p \leq p'$. We say that LTS is *strongly monotone* if and only if every transition of T is strongly monotone.

We also find in [Fin20] a notion more relaxed of monotonicity. We say that an LTS is *monotone* if and only if $\leq$ is upward compatible. Strong monotonicity preserves traces, that is to say that a trace in $\langle LTS, q\rangle$ still exists in any $\langle LTS, q'\rangle$ such that $q \leq q'$.

We say that a trace σ is *repetitive* if and only $\forall q \in \mathtt{Dom}(\sigma), \forall n \in \mathbb{N}, \sigma^n$ is a trace in the LTS paired with q.

Lemma 2. *Let's LTS be strictly monotone and σ be a trace, if $\exists\ q,\ q' \in Q$ such that $q \leq q'$ and $q \xrightarrow{\sigma} q'$ then $\forall n \in \mathbb{N},\ \sigma^n$ is a trace of $\langle LTS, q\rangle$ and σ is repetitive.*

3.9.2. Liveness, Starvation, and Fairness

A transition t is *live* if and only if $\forall q \in RS,\ \exists\ q' and\ q" \in RS$ such that q' is reachable from q and $q' \xrightarrow{t} q"$, or equivalently, if and only if $\forall q \in RS,\ \exists\ \sigma \in T^*$ such that $q \xrightarrow{\sigma t}$.

We say that the pair $\langle LTS, q_0\rangle$ is live if and only if every transitions of T are live.

Liveness is an important property for a transition t. It says that whatever is the progression of the model, it will always be possible to execute t. This does not prevent the existence of infinite progression where t never occurs. This possibility leads to the notions of fairness and starvation.

The term starvation comes from a famous case study called the dining philosophers by Dijkstra described in [Dij71] or in [Hoa15]. In [CM88] p9, we find the following definition of fairness: "every statement is selected infinitely often".

3.10. Petri Nets and a handful of derived models

In order to support an algebraic invariant calculus, dependencies between transitions and state variables need to be more precise than with TS or LTS. We call *topology* the set of information referring to dependencies between transition and state variables in the same informal fashion as the notion of structure. A topology must present enough information for building RS up. We find this topology in Petri Nets. In [Pet96], Petri extended much further the notion of topology, however, we will limit ourselves to Petri Nets as defined in this section.

Petri Nets are among the very first conceptual model to present a topology under the format of a bipartite graph between transitions and state variables.

3.10.1. Petri Nets (PN)

Petri Nets have been introduced by C. A. Petri circa 1962 [13]. Here, we give definitions that are similar to definitions in [Bra82] or in [GV03]. It can be pointed out that the definition of *semi-transition systems* in [Lie76a] is equivalent.

A *Petri Net* is a tuple $PN = \langle P, T, Pre, Post\rangle$, where P is a finite set of *places*, T a finite set of *transitions* such that $P \cap T = \varnothing$. A transition t of T is defined by its $Pre(\cdot, t)$ and $Post(\cdot, t)$ *conditions* [14]: $Pre : P \times T \to \mathbb{N}$ is a function providing a weight for pairs ordered from places to transitions, $Post : P \times T \to \mathbb{N}$ is a function providing a weight for pairs ordered from transitions to places.

Extensive definitions, properties, and case studies can be found in particular in [Bra82, GV03].

In [Pet96], P and T can be infinite; however in this note, we will consider P and T finite which is the common convention. d will denote the cardinal number of P, m the cardinal number of T.

The dynamic behavior of Petri Nets is modeled via markings. A *marking* q is a function from P to $\mathbb{N}$. When $q(p) = k$, it is often said that the place p contains k *tokens*. A *marked Petri Net* is a pair $\langle PN, q_0\rangle$ where PN is a Petri Net and q_0 its initial marking.

We dynamically progress from one state q to the next q' by *enabling* [15] a

[13]Peruse the book cited in [Mem19] for understanding the influence of C. A. Petri in this research community.

[14]We use here the usual notations: $Pre(\cdot, t)(p) = Pre(p, \cdot)(t) = Pre(p, t)$ and $Post(\cdot, t)(p) = Post(p, \cdot)(t) = Post(p, t)$.

[15]the words 'firing', 'occurring', or 'executing' are often used in the literature.

transition t from q if and only if its preconditions (Pre) are satisfied i.e.:

$$q \geq Pre(\cdot, t) \tag{1}$$

In other words, in a PN we have: $\texttt{Dom(t)} = \{q \in Q \mid q \geq Pre(\cdot, t)\}$.

We let the reader verify that regarding property 1 and for any PN, P2 holds but not P1.

We say that t is *enabled* in q. Then, q' can be computed from q with the following equation:

$$q' = Post(\cdot, t) - Pre(\cdot, t) + q \tag{2}$$

A marked Petri Net $\langle PN, q_0 \rangle$ can be often associated with a particular class of Labeled Transition Systems $\langle S, T_{TS}, \rightarrow \rangle$ paired with the same initial state q_0 such that:

Lemma 3. *For any marked PN $\langle \langle P, T_{PN}, Pre, Post \rangle, q_0 \rangle$ we can associate an LTS $\langle S, T_{TS}, \rightarrow \rangle$ with the same initial state q_0 such that: $T_{TS} = T_{PN}$, $S = RS(PN, q_0)$, and $\rightarrow$ is such that: $(s, t, s') \in \rightarrow$ if and only if $s \geq Pre(\cdot, t)$ and $s' = Post(\cdot, t)) - Pre(\cdot, t) + s$.*

This association is non injective: several marked Petri Nets may be associated with the same LTS (since loops can be added to a Petri Nets without modifying its behavior). A similar association can be found in [SSM03]. In this note, in order to follow the idea of the lemma, places will model state variables and transitions will model atomic actions described by equations over these variables (see equation 2).

From this analogy, we can transfer the definitions of reachability set and home space to Petri Nets and other related conceptual models, a state being called a *marking*, state variables being called *places*.

A Petri Net is often represented by an *incidence matrix* over $\mathbb{Z}$ directly derived from Pre and $Post$ considered as matrices of dimension $P \times T$ with entries in $\mathbb{N}$:

$$C = Post - Pre \tag{3}$$

It must be pointed out that C does not take into account loops between a place p and a transition t (we have a loop between p and t if $Pre(p,t) \times Post(p,t) > 0$, in particular If $Pre(p,t) = Post(p,t)$ then $C(p,t) = 0$ as if there would be no connection (i.e. dependency) between p and t). In PNs, a loop between p and t is nevertheless useful since it can model a test on a state variable p constituting a pre-condition to execute t without modifying the value of p.

As for LTS, a trace or a sequence of transitions r is associated with a vector $\overrightarrow{r}$ such that $\overrightarrow{r_i}$ is the number of occurrences of the transition t_i in the sequence r. $\overrightarrow{r}$ is sometimes called the Parikh vector of r since r can be seen as a word of T^* (by lemma 3 a Petri Net can be seen as a Labeled Transition System which in turn can be seen as an automata!). If from q_0, we reach q by enabling the sequence r then we can compute q with the equation:

$$q = C\overrightarrow{r} + q_0 \tag{4}$$

This equation is often referred to as the *state equation* in reference to control theory.

First, let's recall that the matrix C lost possible loops, second moving from r to $\overrightarrow{r}$ looses all information about the order in which the transitions are enabled. Therefore, one should not expect to deduce everything about the behavior of the PN from this single equation. It is nevertheless legitimate to start any NP analysis by studying the equation of state for at least three reasons. First, one can still deduce many properties (as we will see in the examples of the following sections) Second, the complexity of other algorithms that would bring a more complete analysis is often exponential. Finally, the information provided by the study of the state equation can allow simplifying more complex algorithms by pruning away states that can be proven unreachable because they would contradict the state equation.

Since P and T are disjoint, the functions Pre and $Post$ define a bipartite graph between P and T. Petri Nets have inherited a pictorial representation which has not been a minor factor in their success compared to other similar models such as Vector Addition Systems (VAS) (introduced in [KM69] and used in [Ler09] to solve the reachability problem). Transitions are represented by rectangle (or bars) and places by circles (see Figure 9 or 12 for examples). If $Pre(p,t) = k > 0$ then the graph will have an edge from p to t labeled by k. If $Post(p,t) = k > 0$ then the graph will have an edge from t to p labeled by k. We thus formed a bipartite graph: there is no edge neither between any two transitions nor between two places.

If for a place p, $q(p) = n$ we will say that p contains n *tokens* represented by the letter n (by the value of n when this value is known or by n dots when n is sufficiently small) inside the circle representing the place p (see Figure 5 for a first example, Figures 9 and 11 for examples with parameters). As the Petri Net evolves from marking to marking, the number of tokens in places evolves dynamically, this is often denoted by the *token game* of a given Petri Net.

3.10.2. About other models derived from Petri Nets

Often, researchers or engineers would find Petri Nets too low level to describe complex systems and would find Petri Nets almost as complicated to handle as Turing machines in terms level of abstraction if it was not for their handy graphical aspect. Indeed, we know since the reachability problem has been solved for Petri Nets with finite sets of places and transitions (see [Ler09] for an elegant proof using VAS (i.e. PN with no loops [16])) that Petri Nets are strictly less powerful in terms of algorithmic description than Turing machines.

Stressing the need for more concise conceptual models providing a higher level of abstraction, many (maybe too many) proposals were introduced to enrich the token game. These conceptual models are built from PNs using two methods.

First, they are making the enabling rule more elaborate, using for instance, inhibitor arcs (to be able to test that the marking of a place is null), priorities between transitions, time (introduced in [Ram73]), or probabilities.

Second, they are making token types more complex. One of the most famous

[16] This slight difference is already noticed in [Lie76b] later in [Val81]

extension is called *color* [JR91, GV03]: a place p can take its values in $\mathbb{N}^{\mathcal{C}}$ where $\mathcal{C}$ is an enumeration of colors potentially infinite. When $\mathcal{C}$ is finite, it is possible to "unfold" p to get a traditional Petri Net. Other authors transform tokens into letters (introduced in [Mem83, FM83]), predicates [Gen87], or abstract data types [VM84]. At last, let us mention the self-modifying Petri Nets [Val78] where the enabling rule itself dynamically evolves depending on a linear combination of the current state (expanded with a constant). Self-modifying Petri Nets include a number of derived models; they allow very concisely modeling intricate enabling rules supporting the description of complex and flexible systems such as collaborative processes [GAH$^+$93].

As long as they retain a bipartite topology between places and transitions and define a token game of some sort, these extensions keep contributing to forming the wide family of Petri Nets variants.

4. What exactly is an invariant?

4.1. Towards a definition

“In mathematics, an invariant is a property of a mathematical object (or a class of mathematical objects) which remains unchanged after operations or transformations of a certain type are applied to the objects.” [17] In our case, a “mathematical object” would be a property over state variables; and “operations or transformations” would be transitions.

More intuitively, each time you start a sentence by a phrase such as *’is constantly’*, *’from now on’*, *’must always’*, *’for all possible situation’*, *’must never’*, *’is impossible under any circumstances’*, or *’never will’* you can suspect that your sentence can be associated with an invariant. However, the use of words such as *’almost’*, *’likely’*, or *’probable’* can change everything and let you suspect that your sentence will almost surely not be considered as an invariant. Actually, if you start to interleave these phrases, a reader may have some hard time to determine what exactly the resulting statement is meant. For instance, *’is almost constantly’* could sound like an oxymoron and certainly does not suggest an invariant property. This may explain why it is so uneasy to capture the notion of invariant with accuracy.

In computer science or in software engineering, an invariant is a system behavioral property or relationship the value of which does not vary over time, over the evolution of the said system. It can be a property over states of the system under study; it can also be a sequence of actions that happens regularly in a predictable manner within a well defined context.

In system modeling, invariants will be used to describe behavioral requirements or specifications as for instance, in [Hoa15]. In programming, invariants will be used to take care of system exceptions which are often used to support the detection of a possible physical failure or a software bug and provide valuable guards (preventing from an unwanted event leading to an erroneous state)

[17]https://en.wikipedia.org/wiki/Invariant_(mathematics)

or alerts (detecting an unwanted event requiring the execution of an exception). Often, the concepts of constraint, property, requirement, assertion, or exception will be described by using an invariant, especially in the context of the specification of a parallel system.

Invariants have a constant truth value that is always satisfied for any evolution of the Transition System (see [Kel76] for a general definition) modeling a system or a concurrent program under study. An *invariant* is a property over the state variables true for all elements of the reachability set (in RS). This is indeed different from a property true in Q i.e. for all states (reachable or not) which would be close to be a tautology in regard to the model behavior.

Invariants can be described and verified in a modal logic, particularly in a temporal logic (see the seminal work by A. Pnueli in [Pnu77] or the definition of a Computation Tree Logic (CTL) in [HT87] or CTL* in [EH86]).

Concerning Petri Nets, these properties or relationships are expressed or deduced by using markings, places, and transitions. The bipartite graph which supports the topology of the system under study, connecting places and transitions can be seen as a graphical view of a system of Diophantine equations. Together with the *"token game"*, which in turn models the dynamic evolution of the system under study, we have two distinctive elements which make Petri Nets so unique and popular. As long this topology remains in any derivation of a Petri Nets, we believe that the notion of invariant will also remains.

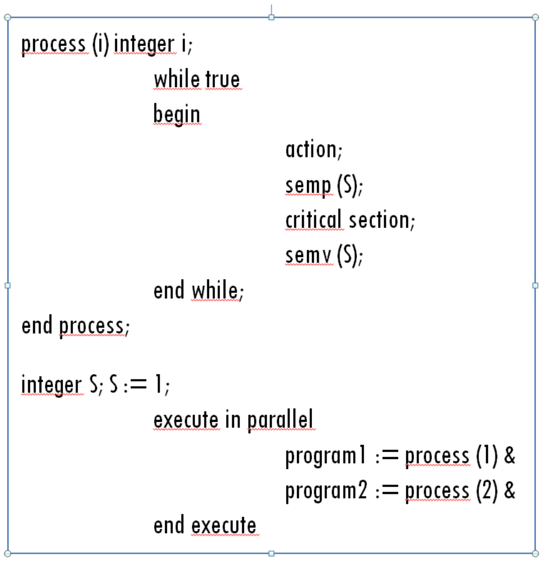
In a way, considering markings and tokens, a vast family of invariants can also be seen as constant functions over the distribution of tokens in a Petri Net independently of the sophistication of its enabling rule or the complexity of the structure of its tokens.

4.2. Mutual exclusion: a classic example of invariant

A classic example of invariant is related to the mutual exclusion mechanism (see [Ray86] for many elaborated discussions and solutions on the topic) between activities known as critical sections of two or more concurrent programs. A *critical section* is a piece of code handling a set of resources in a way that must be unambiguous [Dij71]; this can be achieve by excluding other pieces of code from simultaneously modifying or manipulating the considered shared resources. In other words, the concurrent programs must never[18] be active in their respective `critical section` at the same time.

4.2.1. Two programs in mutual exclusion

In the example Figure 2, we consider two programs sharing a common resource. The mutual exclusion between the critical sections is ensured by a Dijkstra semaphore `S` [Dij71] and the common resource will be abstracted away.



```
process (i) integer i;
        while true
        begin
                action;
                semp (S);
                critical section;
                semv (S);
        end while;
end process;

integer S; S := 1;
        execute in parallel
                program1 := process (1) &
                program2 := process (2) &
        end execute
```

Figure 2: `Program1` and `Program2` are two instances of the process(i) and are concurrently running sharing a common resource guarded by a semaphore `S`. `S` is initialized to 1 to guarantee the mutual exclusion between the `critical section` of each program. We assume that there are no dependencies (i.e. shared resources) between an initial piece of code called `action` and the `critical section`, in particular that only the `critical section` handles the common resource protected by the semaphore.

After running an `action` which has no use of the common resource, a pro-

[18]our reader would have recognized one of the phrases of Section 4.1 leading to an invariant definition

gram can enter in its `critical section` only by executing the indivisible primitive `semp` of the semaphore `S`. Initialing `S` to z will not authorize more than z execution of `semp` in a row. To ensure the mutual exclusion between the two programs, `S` must be initialized to 1.

A program executing `semp(S)` will be put on hold (in a sleeping mode in a queue for instance) if $\texttt{S} \leq 0$ else if $S > 0$ then $\texttt{S} := \texttt{S} - 1$ and the program can access its `critical section`. This must be permitted by the semaphore only if the other program is not itself in its `critical section`.

The program exits from its `critical section` by executing `semv(S)`. A program executing `semv(S)` will increment `S` by 1 and will wake up any other program put on hold by a `semp(S)`, granting one of them an access right to enter in their own `critical section` by successfully executing again `semp(S)`. To guarantee the mutual exclusion, we must be sure that $\texttt{S} < 2$ for all reachable state. Let us model the two programs by an LTS to prove that the following property is an invariant: "for all reachable state, if one of the two programs is its critical section then the other one is not."

4.2.2. Modelization with an LTS

Let us model this example by a labeled transition system $LTS = \langle Q, T, \rightarrow \rangle$. First, each Process(i) is associated with one state variable: its ordinal counter OCi. Usually, an ordinal counter points to the memory address of the next statement to be executed. In this example, the ordinal counter OCi varies in $Instr = \{1, 2, 3, 4\}$ with the following meaning:

1 : when process(i) is ready to execute `action`,

2 : when process(i) is ready to execute `semp`,

3 : when process(i) is ready to execute `critical section`,

4 : when process(i) is ready to execute `semv`.

The set $\mathcal{SV}$ is composed of 3 state variables: $\mathcal{SV} = \{OC1, OC2, S\}$ one state variable per ordinal counter and one state variable for the semaphore. The statement "$S = S - 1$" being guarded by testing if "$S > 0$" we can conclude that the state space $Q = Instr \times Instr \times \mathbb{N}$; the initial state is $q_0 = (1, 1, 1)$.

The set of transitions T is matching the set of statements of the two programs:

$T = \{A1, Semp1, CS1, Semv1, A2, Semp2, CS2, Semv2\}$

where each program is represented by 4 transitions with the following definitions for the relation $\rightarrow$:

• Ai for `action`, with $\forall x, y \in Instr, \forall z \in \mathbb{N}$,

$A1 : (1, y, z) \rightarrow (2, y, z)$ and $A2 : (x, 1, z) \rightarrow (x, 2, z)$,

• $Sempi$ for `semp(S)` with $\forall x, y \in Instr, \forall z \in \mathbb{N}^+$,

$Semp1 : (2, y, z) \rightarrow (3, y, z - 1)$ and $Semp2 : (x, 2, z) \rightarrow (x, 3, z - 1)$,

• CSi for `critical section` with $\forall x, y \in Instr, \forall z \in \mathbb{N}$,

$CS1 : (3, y, z) \rightarrow (4, y, z)$ and $CS2 : (x, 3, z) \rightarrow (x, 4, z)$,

• $Semvi$ for `semv(S)` with $\forall x, y \in Instr, \forall z \in \mathbb{N}$,

$Semv1 : (4, y, z) \rightarrow (1, y, z + 1)$ and $Semv2 : (x, 4, z) \rightarrow (x, 1, z + 1)$,

The knowledge of the overall program and the definition of $\mathcal{SV}$ make possible the construction of the reachability set RS of this LTS. Actually, starting from q_0, its labeled reachability graph can easily be constructed as shown in Figure 3. From q_0 we can only enable the transitions Ai to reach the states (2,1,1) (by enabling $A1$) and (1,2,1) (by enabling $A2$) and so on. We can observe that RS is finite and that there is no state $q \in RS$ such that $q = (3, 3, z)$ where $z \in \mathbb{N}$. Therefore, the two programs are never executing their `critical section` at the same time and our invariant is proven.

4.2.3. Modelization with a PN

We also can model our example with a Petri Net directly borrowing its representation from the LTS we just defined: with the same set T of transitions and since we know $\mathcal{SV}$ and since $Instr$ is finite, we will have one place for each value of the ordinal counter of each program. The left hand side of Figure 4 is showing how `program1` can be modeled by a Petri Net. Moreover, noticing that the transitions $A1$ and $CS1$ have only the ordinal counter for input and output, we propose to reduce the places $OC_{1,1}$ and $OC_{1,2}$ with the transition $A1$ into a single place A and similarly reduce the places $OC_{1,3}$ and $OC_{1,4}$ with

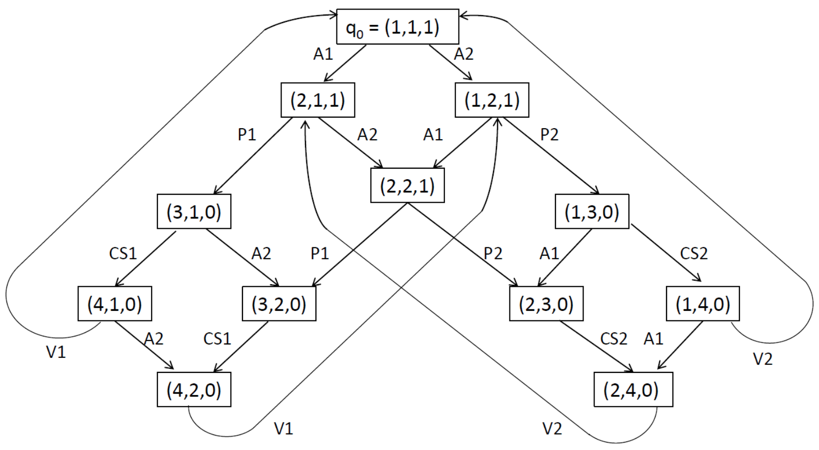


Figure 3: Labeled Reachability Graph (LRG) of the LTS modeling the two concurrent porgrams in mutual exclusion of figure 2. Each vertex is labeled by a state of RS, each edge is labeled by a transition of T. We can observe that $q(\mathtt{S}) < 2$ for all reachable state: the semaphore is accomplishing its mission in preventing the execution of `semp` twice in a row

the transition $CS1$ into a single place B. Places A and B are sometimes called *macroplace* (see chapter 15 in [GV03] or chapter 4 in [Bra82] for more details on reduction rules). We get the Petri Net on the right hand side of Figure 4. The semaphore is now modeled by a single place S to obtain the Petri Net Figure 5.

In the Petri Net described in Figure 5, the two concurrent programs are represented by the two subgraphs within dotted lines. The semaphore primitives `semp` and `semv` by the transitions $Semp_1$, $Semp_2$ and $Semv_1$, $Semv_2$ respectively. The initial marking q_0 has one token in A (`program1` is not in its `critical section`), one token in D (`program2` is not in its `critical section`), and one token in **S** (the semaphore is ready to let pass one and only one program in its `critical section`: either $Semp_1$ or $Semp_2$ will be enabled). The fact that a `program`$_i$ is active in its `critical section` is modeled by the presence of a token in the places B or E respectively. The mutual exclusion between the two programs can be expressed in various different ways, and the corresponding invariant can be rewritten under at least the four following

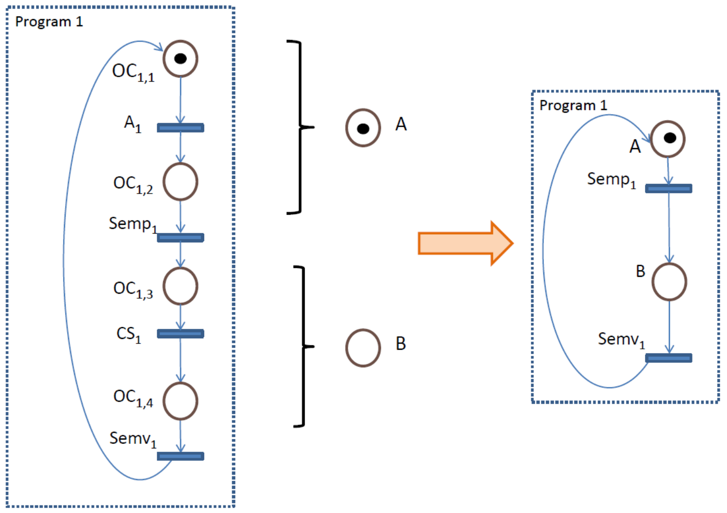


Figure 4: On the left, `program1` is modeled by a Petri Net where all the elements of the LTS of Section 4.2.2 are represented. Then, it is reduced to a smaller one behaving similarly in regard to mutual exclusion. A token in the place A means that the program 1 is not in its `critical section` and may be executing another piece of code (i.e. `action`). A token in the place B means that the program 1 is in its `critical section`. We must not simplify the Petri Net any further since we do want to observe the mutual exclusion with `program2` through the connections via the transitions $Semp_1$, $Semp_2$, $Semv_1$, $Semv_2$, and the place **S** .

equivalent forms [19]: for any marking q reachable from the initial marking q_0:

(ME) $q(B) \times q(E) = 0$

(ME') $(q(B) = 0) \oplus (q(E) = 0)$

(ME") $q(B) + q(E) \leq 1$

(ME"') $q(S) < 2$

We let the reader prove the equivalency of these four statements given the Petri Net paired with the initial marking of Figure 5. We will not study or utilize rewriting systems in this note (there are many books on this topic and already suggested [DJ90]). We just want to suggest to the reader to consider rewriting his problem or his model in different equivalent ways especially when

[19]where $\oplus$ stands for the xor Boolean operator.

using a verification tool such as a model checker or a prover which can be very sensitive to the way a formula is written (as a trivial example, the order in which commutative terms of a Boolean disjunctive normal form are written and evaluated) and can very well develop a proof with very different performance.

This will easily be proven in Section 5.2.1 without exploring every state of RS. We let the reader build the labeled reachability graph of this Petri Net and compare it with the LRG of Figure 3.

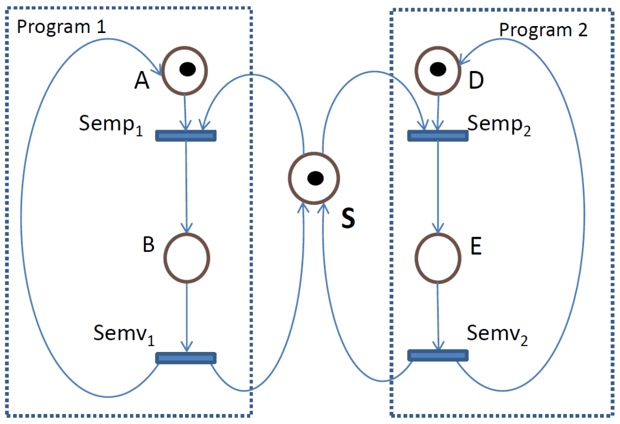


Figure 5: Program 1 and Program 2 must be in mutual exclusion. For instance, as they are sharing a common resource guarded by a semaphore **S**, they must not access (and modify) this shared resource at the same time to ensure its consistency. The state of the shared resource is being consistent with the state of **S** and does not need to be described.

We will discuss further an extension of this possible solution in Section 7. Other synchronization mechanisms for ensuring mutual exclusion between critical sections of concurrent processes can be found in the literature (see for instance, [RT19] for many different examples with various degrees of sophistication or [MVD03] for three interesting case studies using Petri Nets).

5. Invariants, conservative components, and semiflows

One straightforward method to prove that a proposition is an invariant would be by exploring all reachable state of the model, especially by using a model checker, a powerful tool that automatizes and optimizes such an exploration (see [MOSS99] for a tutorial on model checking, see [Wol19b] for Petri Net model checking). Symbolic state-space exploration can also be used in the presence of parameters in the model. Other methods would be for instance, by induction (see [Kel76] for an example of an induction proof in a Transition System). However, an important question is not only to prove that a given proposition is an invariant (which is no small undertaking), but also to design algorithms able to discover as many of them as possible with as little guidance as possible and ultimately, understand which ones are the most potent.

We first associate to any property or relationship PR, the set of states $supp(PR)$ for which PR is satisfied. $supp(PR)$ is said to be the *support* of PR. Likewise, an invariant I is associated with a set $supp(I)$ of all states satisfying the invariant property or relationship (i.e., for which the invariant is true): I will be an invariant if and only if $RS \subseteq supp(I)$ [20]. In other words, an invariant can be used to describe (most of the time concisely) a superset of the reachable states of a model. The smaller the support the more potent the corresponding invariant will be.

The set of states $supp(I)$ associated with an invariant I is a home space since it includes the entire reachability set.

5.1. Invariant calculus

If I and J are two invariants associated with their respective supports $supp(I)$ and $supp(J)$, then it is easy to define two invariants K_u and K_i such that $supp(K_u) = supp(I) \cup supp(J)$ and $supp(K_i) = supp(I) \cap supp(J)$. A similar observation can be found in [CM88]. The intersection is stable over the set of supports of invariants because the reachability set is included in any support of invariant; this property is important since we are looking at manipulating sets as small as possible. Let's recall that the intersection is not stable for home

[20]Here, it is worth pointing out that it is possible to have $supp(I) \setminus Q \neq \varnothing$ depending on the definition of Q

spaces (property 2 section 3.4). It is also worth pointing out that it is easy to combine invariants in various ways and to generate many of them. It is therefore important to understand whether invariants are organized and how to develop some computational rules over invariants relative to a given Transition System or a given Petri Net.

These observations led to look for the smallest set of states associated with invariants, or at least to look for the smallest ones that can be easily handled from an algebraic point of view.

If an invariant I can be associated with its eval function $eval_I$ over states, then from equation (2) in section 3.10.1 we have:

$$eval_I(q') = eval_I(Post(\cdot, t) - Pre(\cdot, t) + q) \quad \forall t \in T \tag{5}$$

Moreover, if $eval_I$ has the good taste to be a morphism that preserves addition, then we may rewrite our system of equations to obtain:

$$eval_I(q') = eval_I(Post(\cdot, t)) - eval_I(Pre(\cdot, t)) + eval_I(q) \quad \forall t \in T \tag{6}$$

Since I is an invariant, $eval_I(q') = eval_I(q)$ therefore, $eval_I$ must verify the following system of equations:

$$eval_I(Post(\cdot, t)) = eval_I(Pre(\cdot, t)) \quad \forall t \in T \tag{7}$$

5.2. Semiflows Calculus

Until now, all these definitions have been pretty general and can be applied to any Labeled Transition System model, in particular, any extension of the Petri Net conceptual model such as the ones that can be found in [Val81], [JR91], [FS01], or [GV03].

Semiflows calculus is possible as soon as states and transitions are described with state variables. Any subset of transitions allows deducing a system of equations with state variables which must not be modified outside the considered subset of transitions. Then, a solution to this system of equations will be called a semiflow.

In many Petri Nets papers, a specific class of invariants is considered. They are sometimes unduly called *linear invariants* because they are associated with an integer weight function f over the set P of places. With regard to a Petri Net PN, these invariants have the following form:

$$eval_I(q) = f^\top q = eval_I(q_0) = f^\top q_0 \quad \forall q \in RS(PN, q_0) \tag{8}$$

where $f^\top q = \sum_{p \in P} f(p)q(p)$ can be seen as the scalar product of f and q seen as vectors in $\mathbb{Z}^d$.

Y.E. Lien [Lie73] and then independently, K. Lautenbach, and H. A. Schmid [LS74] were among the first ones, if not the first ones, to describe such a concept for Petri Nets. The algebraic calculus underneath this concept was first described in [Mem77] followed by many other publications such as [Sif78, STC96, Mem23]. The vast corpus of linear algebra results and algorithms can be applied to analyze Petri Nets and compute semiflows.

These weight functions can be computed by solving the following homogeneous system of m Diophantine equations directly derived from equation (7):

$$f^\top Post(\cdot, t) = f^\top Pre(\cdot, t) \quad \forall t \in T \tag{9}$$

The system of equations (7) can already be found in [Mem19] while the system 9 is well-known in the literature [Bra82, STC96, GV03]. This system of equations is fundamental; it describes some constraints over the topology of the Petri Net independently of any initial state and expresses a law of conservation over the net and its evolution. It says that during the dynamic evolution of the Petri Net, a function of token distribution remains constant: what is consumed and needed to enable t is equal to what is produced after enabling t modulo the weight function f. This is a law similar to Kirchhoff's first law on the conservation of current flows but only applied on transition nodes i.e. half of the nodes of the bipartite graph representing PN. This explains why such functions over P (or equivalently vectors in $\mathbb{Z}^d$) have been baptised *semiflow* in [Mem77].

Several algorithms were independently developed to compute a generating set of semiflows [Tou81] or later in [MS82, AM82]. All of them can be considered as variations of Farkas or Fourier algorithms related to integer linear programming and convex geometry (see [CS89] for a comparative study or [Sch87] for the underlying mathematical theory).

Of course, any linear combination of semiflows is still a semiflow. The set $\mathcal{F}$ of semiflows with integer coefficients is a module since $\mathbb{Z}$ is a ring. In subsection 5.3.2, we will examine the subset $\mathcal{F}^+ = \{f \in \mathcal{F} \mid \forall p \in P, f(p) \geq 0\}$ of $\mathcal{F}$ and see why this subset is of particular interest. However, it is still possible to characterize this set with a generating set by considering the notion of support for semiflows. If f is a semiflow then $\|f\|$ denotes the *support of f* and $\|f\| = \{x \in P \mid f(x) \neq 0\}$. Often, $\|f\|$ is also called *conservative component* [Lie76b] or [Bra82].

5.2.1. Semiflows and Invariants in the mutex example

The Petri Net of Figure 5 has 3 semiflows f_1, f_2, and *sem*:

f_1 such that $f_1(A) = f_1(B) = 1$, $f_1(p) = 0$ for any other place p.

f_2 such that $f_2(D) = f_2(E) = 1$, $f_2(p) = 0$ for any other place p.

sem such that $sem(B) = sem(E) = sem(\mathbf{S}) = 1$, $sem(p) = 0$ for any other place p.

It is easy to verify that f_1, f_2, and *sem* are minimal of minimal support. f_1 is associated with Program 1. Its associated invariant means that there is always only one token in the support $\{A, B\}$ of f_1 i.e. either in A or in B. Similarly, f_2 is associated with Program 2. Its associated invariant means that there is always only one token between D and E. The token in $\{A, B\}$ can be interpreted as modeling the ordinal counter of Program1; similarly, the token in $\{D, E\}$ can be interpreted as modeling the ordinal counter of Program2. The semiflow *sem* is associated with the semaphore **S** and its associated invariant is:

$sem^\top q = sem^\top q_0$ in other words: $sem(B) + sem(E) + sem(\mathbf{S}) = 1$ for any reachable marking q from q_0.

It allows to straightforwardly deduce that (ME) defined in Section 4.2.3 is an invariant since there is at most one token in $\{B, E\}$. □

5.3. Invariants pave the way to boundedness, sometimes liveness or fairness

In regard to Petri Nets, invariants are interesting not only because they express a property that is looked for itself (for instance, to comply with the system specification) but also because they straightforwardly allow the exclusion of a large number of markings that cannot be reached from the initial marking without violating the invariant. In other words, if a marking does not satisfy an invariant then it is unreachable. This important aspect is at the source of pruning techniques and is critical to analyze and prove many important behavioral Petri Net properties such as liveness, boundedness, or even fairness which otherwise must be analyzed by developing the reachability or one coverability graph of the Petri Net under consideration (see [KM69] and [Fin93] for a minimal coverability graph). Many of these results are described and proven in [Bra82].

5.3.1. Home spaces, semiflows, and liveness

Semiflows are intimately associated with home spaces and invariants and can greatly simplify the proof fundamental properties of Petri Nets (even including parameters as in [BEI$^+$20]) such as safeness, boundedness, or more complex behavioral properties such as liveness. Let us provide three properties supporting this idea.

Property 6. A transition t is live if and only if `Dom(t)` is a home space.

Moreover, if `Dom(t)` is a home space then `Im(t)` is also a home space.

This relationship is further supported when considering the following property regarding home states:

Property 7. Let M be a model and q_0 is a home state then:

any transition that is enabled from q_0 is live,

more generally,

a transition is live if and only if it appears as a label in $LGR(M, q_0)$.

This can easily be proven directly from the definition of liveness section 3.9.2 and the property 5 about home states . □

Given an initial state q_0, each semiflows can be associated with an invariant which in turn can be associated with a home space. In other words, if $f \in \mathcal{F}$, then $HS(f) = \{q \in Q \mid f^\top q = f^\top q_0\}$ is a home space.

Property 8. If $f \in \mathcal{F}$ then: $\forall \lambda \in \mathbb{Q} \setminus \{0\}, HS(\lambda f) = HS(f)$,
$\forall g \in \mathcal{F}, HS(f) \cap HS(g) \subseteq HS(f+g)$

If $q \in HS(f) \cap HS(g)$, then $f^\top q = f^\top q_0$ and $g^\top q = g^\top q_0$, so $(f+g)^\top q = (f+g)^\top q_0$, therefore, $q \in HS(f+g)$ □

These three properties provide us with a methodology to analyze and prove that a subset of transitions are live. From a set of invariants, we can define a first home space HS that concisely describe how tokens are distributed over places. From this token distribution, we can analyze what transition are enabled in order to prove that a specific given marking q (q_0 being the usual case) is always reachable from any element of HS. When this is possible, it can easily be deduced that q is a home state. Then, it may be possible using property 7 to prove which transition are live and whether the Petri Net is live or not. This will be illustrated later with a few examples in section 7.

5.3.2. Semiflow basic properties

Solutions of the system of equations (9) have their coefficients in $\mathbb{Z}$ in general. However, the most interesting semiflows from a behavioral analysis point of view are defined over natural numbers instead of integers. This can be seen through the three following properties. First, we can define the *positive and negative supports* of a semiflow $f \in \mathcal{F}$ as:

$\|f\|_+ = \{p \in P \mid f(p) > 0\}$,

$\|f\|_- = \{p \in P \mid f(p) < 0\}$,

with $\|f\| = \|f\|_- \cup \ \|f\|_+$.

We can then rewrite the system of equations (8):

$f^\top q = \left|\sum_{p \in \|f\|_+} f(p)q(p)\right| - \left|\sum_{p \in \|f\|_-} f(p)q(p)\right| = f^\top q_0$

As we can see, it is a differential between the weighted number of tokens in the places of the positive support and the weighted number of tokens in the places of the negative support of f that remains constant during the evolution of the Petri Net. A first general property can be immediately deduced assuming that the initial state $q_0 \in \mathbb{N}^d$:

Property 9. For any semiflow $f \in \mathcal{F}$, $\exists p \in \|f\|_+$ not bounded if and only if $\exists p \in \|f\|_-$ not bounded.

Of course, if $\|f\|_- = \varnothing$ then $f \in \mathcal{F}^+$ and $\|f\|$ is necessarily bounded. More generally, considering a weighting function f over P being defined over non-negative integers, it can be easily proven [Mem78]:

Property 10. If $f \geq 0$ and $f^T Pre(\cdot, t) \geq f^T Post(\cdot, t) \forall t \in T$, then the set of places of $\|f\|$ is structurally bounded [21]. Moreover, the marking of any place p of $\|f\|$ has an upper bound:
$q(p) \leq \frac{f^T q_0}{f(p)}, \quad \forall q \in RS(PN, q_0)$.

If $f > 0$ then $\|f\| = P$ and the Petri Net is also structurally bounded. The reverse is also true: if the Petri Net is structurally bounded, then there exists a strictly positive solution for the system of inequalities above (see [Sif78] or [Bra82]). This property is indeed false for a semiflow that verifies the system but would have at least one negative element and constitutes a first reason for particularly considering weight functions f over P being defined over non-negative integers including $\mathcal{F}^+$.

The following corollary can directly be deduced from the fact that any semiflow in $\mathcal{F}^+$ satisfies property 10:

Corollary 1. *For any place p belonging to a least one support of a semiflow of $\mathcal{F}^+$, an upper bound μ can be defined for the marking of p such that:*
$q(p) \leq \mu(p, q_0) = \min_{\{f \in \mathcal{F}^+ \ \mid \ f(p) \neq 0\}} \frac{f^\top q_0}{f(p)} \quad \forall q \in RS(PN, q_0)$

A second reason for particularly considering a semiflow f as being defined over non-negative integers is that the system of inequalities:

[21] following the general definitions given in Section 3.5

$$f^T q_0 \geq f^T Pre(\cdot, t), \quad \forall t \in T \tag{10}$$

becomes a necessary condition for any transition t to stand a chance to be enabled from any reachable marking from q_0, then to be live. In [Bra82], $f^T Pre(\cdot, t)$ is called the *enabling threshold* of t.

At last, the following property can easily be proven true in $\mathcal{F}^+$ and not true in $\mathcal{F}$:

Property 11. If f and g are two semiflows with no negative coefficients, then we have: $\|f + g\| = \|f\| \cup \|g\|$.

This property can already be found in [Mem78, Bra82].

These results have been cited and utilized many times in various applications going beyond computer science, electrical engineering, or software engineering. For instance, they have very recently been used in the domain of biomolecular chemistry relatively to chemical reaction networks [JACB18] which brings us back to the original vision of C. A. Petri when he claimed that his nets could be used in chemistry.

6. Generating sets and minimality

6.1. Generating sets

The notion of generating sets for semiflows is well known and efficiently supports the handling of an important class of invariants sometimes unduly called linear invariants in the literature. Several results have been published starting from the initial definition and structure of semiflows [Mem77] to a large array of applications used especially to analyze Petri Nets [CTSH03, DL16, JACB18, Wol19a].

Minimality of semiflows and minimality of their supports are critical for accelerating analysis compared to using other semiflows. Invariants deduced from minimal semiflows relate to smaller quantities of resources. Furthermore, the smaller the support of semiflows, the more local their footprint. In the end, these two notions of minimality will foster analysis optimization.

6.2. Basic definitions and results

A subset $\mathcal{G}$ of $\mathcal{F}^+$ is a *generating set over a set* $\mathbb{S}$ if and only if $\forall f \in \mathcal{F}^+$ we have $f = \sum_{g_i \in \mathcal{G}} \alpha_i g_i$ where $\alpha_i \in \mathbb{S}$ and $\mathbb{S} \in \{\mathbb{N}, \mathbb{Q}^+, \mathbb{Q}\}$, where $\mathbb{Q}^+$ denotes the set of non-negative rational numbers.

Since $\mathbb{N} \subset \mathbb{Q}^+ \subset \mathbb{Q}$, [22] a generating set over $\mathbb{N}$ is also a generating set over $\mathbb{Q}^+$, and a generating set over $\mathbb{Q}^+$ is also a generating set over $\mathbb{Q}$. However, the reverse is not true and, in our opinion, one source of some of the inaccuracies found in the literature.

The notion of the generating set is strongly related to algebraic concepts especially when it is finite. Let's consider $\mathcal{G}$ a finite generating set of $\mathcal{F}^+$ such that $\mathcal{G} = \{g_1, ...g_k\}$, the following definitions can be recalled.

-If $\mathcal{G}$ is a generating set over $\mathbb{N}$ then

$\mathcal{S}(\mathcal{G}) = \{f \in \mathbb{N}^d \mid f = \sum_{i=1}^{i=k} \alpha_i g_i$ and $\alpha_i \in \mathbb{N}\}$ is a *semigroup* and $\mathcal{F}^+ = \mathcal{S}(\mathcal{G})$.

-If $\mathcal{G}$ is a generating set over $\mathbb{Q}^+$ then

$\mathcal{C}(\mathcal{G}) = \{f \in (\mathbb{Q}^+)^d \mid f = \sum_{i=1}^{i=k} \alpha_i g_i$ and $\alpha_i \in \mathbb{Q}^+\}$ is a *convex polyhedral cone* and $\mathcal{F}^+ = \mathcal{C}(\mathcal{G}) \cap \mathbb{N}^d$. It is interesting to recall a result from [LM89] stating that $\mathcal{F}^+ \neq \{0\}$ if and only if $\mathcal{C}(\mathcal{G}) \neq \{0\}$.

-If $\mathcal{G}$ is a generating set over $\mathbb{Q}$ then

$\mathcal{V}(\mathcal{G}) = \{f \in \mathbb{Q}^d \mid f = \sum_{i=1}^{i=k} \alpha_i g_i$ and $\alpha_i \in \mathbb{Q}\}$ is a *vector space* and $\mathcal{F}^+ = \mathcal{V}(\mathcal{G}) \cap \mathbb{N}^d$. We can extract from $\mathcal{G}$ a basis of $\mathcal{V}(\mathcal{G})$ (see, for instance, [Lan02] p. 85) which also is a generating set of $\mathcal{F}^+$ over $\mathbb{Q}$ since the elements of this basis are in $\mathcal{F}^+$.

The knowledge of one generating set allows a practical computation of the bounds defined in property 8 Section 5.3.1, corollary 1 and property 11 of Section 5.3.2:

Theorem 1. *Let's $\mathcal{E} = \{e_1, ...e_N\}$ be any generating set of $\mathcal{F}^+$.*

- *If $\mathcal{E}$ is over $\mathbb{S}$ then we have:*

[22] Where $\subset$ denotes the strict inclusion between sets.

$\iota = \bigcap_{f\in\mathcal{F}^+} HS(f) = \bigcap_{e_i\in\mathcal{E}} HS(e_i)$,

- *If $\mathcal{E}$ is over $\mathbb{Q}^+$ or $\mathbb{N}$ then for any place p belonging to a least one support of a semiflow of $\mathcal{F}^+$, $\forall q \in RS(PN, q_0)$, we have :*

  $q(p) \leq \mu(p, q_0) = \min_{\{f\in\mathcal{F}^+ \mid f(p)\neq 0\}} \frac{f^\top(q_0)}{f(p)} = \min_{\{e_i\in\mathcal{E} \mid e_i(p)\neq 0\}} \frac{{e_i}^\top(q_0)}{e_i(p)}$,

- *If $\mathcal{E}$ is over $\mathbb{S}$ then we have:*

  $\rho = \left\|\sum_{f\in\mathcal{F}^+} f\right\| = \bigcup_{f\in\mathcal{F}^+} \|f\| = \bigcup_{e_i\in\mathcal{E}} \|e_i\|$

Let's consider $f \in \mathcal{F}^+$ with $f = \sum_{i=1}^{i=N} \lambda_i e_i$ and $q \in \bigcap_{e_i\in\mathcal{E}} HS(e_i)$, then:
$\lambda_i(e_i)^\top q = \lambda_i(e_i)^\top q_0 \ \forall \in \{1, ...N\}$, hence:
$\sum_{i=1}^{i=N} \lambda_i(e_i)^\top q = \sum_{i=1}^{i=N} \lambda_i(e_i)^\top q_0$, then:
$f^\top q = f^\top q_0$, and $q \in HS(f) \ \ \forall f \in \mathcal{F}^+$ therefore,
since $\bigcap_{f\in\mathcal{F}^+} HS(f) \subseteq \bigcap_{e_i\in\mathcal{E}} HS(e_i)$ because $\mathcal{E} \subseteq \mathcal{F}^+$, we have:
$\bigcap_{e_i\in\mathcal{E}} HS(e_i) = \bigcap_{f\in\mathcal{F}^+} HS(f) = \iota$.

For the second item of the theorem, let's consider a state q_0, a place p, and a semiflow f of $\mathcal{F}^+$ such that $f(p) > 0$ and $f = \sum_{i=1}^{i=N} \alpha_i e_i$ where $\alpha_i \geq 0 \ \forall i \in \{1, ...N\}$. Let's consider j such that $1 \leq j \leq N$ and define $\mu_\mathcal{E}$ such that:

$\mu_\mathcal{E} = \frac{{e_j}^\top q_0}{e_j(p)} = \min_{\{e_i\in\mathcal{E} \mid e_i(p)\neq 0\}} \frac{{e_i}^\top(q_0)}{e_i(p)}$, then $\forall i \leq N, \exists \delta_i \in \mathbb{Q}^+$ such that: $\frac{{e_j}^\top q_0}{e_j(p)} = \frac{{e_i}^\top q_0 - \delta_i}{e_i(p)}$. It can then be deduced:

$\mu_\mathcal{E} = \frac{\alpha_j {e_j}^\top q_0}{\alpha_j e_j(p)} = \frac{\alpha_i({e_i}^\top q_0 - \delta_i)}{\alpha_i e_i(p)}$ $\forall i$ such that $e_i(p) \neq 0$, therefore:

$$\mu_\mathcal{E} = \frac{\sum_{\{i \mid e_i(p)>0\}} \alpha_i({e_i}^\top q_0 - \delta_i)}{\sum_{\{i \mid e_i(p)>0\}} \alpha_i e_i(p)}$$

$$= \frac{\sum_{\{i \mid e_i(p)>0\}} \alpha_i({e_i}^\top q_0 - \delta_i) + \sum_{\{i \mid e_i(p)=0\}} (\alpha_i {e_i}^\top q_0 - \alpha_i {e_i}^\top q_0)}{\sum_{\{i \mid e_i(p)>0\}} \alpha_i e_i(p)}$$

$$= \frac{\sum_{\{i|e_i(p)>0\}} \alpha_i {e_i}^\top q_0 + \sum_{\{i|e_i(p)=0\}} \alpha_i {e_i}^\top q_0 - \sum_{\{i|e_i(p)>0\}} \alpha_i\delta_i - \sum_{\{i \mid e_i(p)=0\}} \alpha_i {e_i}^\top q_0}{\sum_{\{i \mid e_i(p)>0\}} \alpha_i e_i(p) + \sum_{\{i \mid e_i(p)=0\}} \alpha_i e_i(p)}$$

since $\sum_{\{i \mid e_i(p)=0\}} \alpha_i e_i(p) = 0$. Then, since $\delta_i > 0$ and $\alpha_i > 0 \ \forall i \leq N$

$$\mu_\mathcal{E} = \frac{\sum_{i=1}^{i=N} \alpha_i {e_i}^\top q_0 - \sum_{\{i|e_i(p)>0\}} \alpha_i\delta_i - \sum_{\{i \mid e_i(p)=0\}} \alpha_i {e_i}^\top q_0}{\sum_{i=1}^{i=N} \alpha_i e_i(p)}$$

$$\mu_\mathcal{E} = \frac{f^\top q_0 - \sum_{\{i|e_i(p)>0\}} \alpha_i\delta_i - \sum_{\{i \mid e_i(p)=0\}} \alpha_i {e_i}^\top q_0}{f(p)} \leq \frac{f^\top q_0}{f(p)}$$

This being verified for any semiflow of $\mathcal{F}^+$, we have: $\mu(p, q-0) = \mu_\mathcal{E}$.

For the third item of the theorem, let's consider $\mathcal{E}$ a generating set over $\mathbb{S}$ then, any semiflow f can be decomposed as follows:

$f = \sum_{\alpha_i>0} \alpha_i e_i + \sum_{\alpha_i<0} \alpha_i e_i$ where $\alpha_i \in \mathbb{S}$. $f \in \mathcal{F}^+$ means that at least one coefficient α_i is strictly positive.

Therefore, applying property 11:

$\|f\| \subseteq \left\|\sum_{\alpha_i>0} \alpha_i e_i\right\| = \bigcup_{\alpha_i>0} \|\alpha_i e_i\| \subseteq \bigcup_{e_i \in \mathcal{E}} \|e_i\|$.

Hence, $\rho = \left\|\sum_{f \in \mathcal{F}^+} f\right\| = \bigcup_{e_i \in \mathcal{E}} \|e_i\|$ □

This theorem means that these three parameters ι, μ and ρ can be computed with the help one generating set and furthermore that their values are independent of the chosen generating set. The third part of this theorem means that if $\mathcal{E} = \{e_1, ...e_N\}$ is any generating set of a given Petri Net PN then $\bigcup_{e_i \in \mathcal{E}} \|e_i\|$ is also the maximal support of PN.

6.3. Minimal supports and minimal semiflows

The fact that there exists a finite generating set over $\mathbb{N}$ is non trivial. This result was proven by Gordan circa 1885 then Dickson circa 1913. Here, we directly rewrite Gordan's lemma [AB86] by adapting it to our notations.

Lemma 4. ***(Gordan circa 1885)*** *Let $\mathcal{F}^+$ be the set of non-negative integer solutions of the system of equations (9). Then, there exists a finite generating set of vectors in $\mathcal{F}^+$ such that every element of $\mathcal{F}^+$ is a linear combination of these vectors with non-negative integer coefficients.*

The question of the existence of a finite generating set being solved for $\mathbb{N}$, it is necessarily solved for $\mathbb{Q}^+$ and $\mathbb{Q}$.

Several definitions of the notion of minimal semiflow were introduced in [STC98] p. 319, in [CST03] p. 68, [KJ87], [CMPAW09], or in [Mem78, Mem83]. It can be confusing to look into these in details. In light of this, we propose to consider only two basic notions in order theory: minimality of support with respect to set inclusion and minimality of semiflow with respect to the componentwise partial order on $\mathbb{N}^d$ since the various definitions we found in the literature as well as the results of this note can be described in terms of these sole two classic notions.

A non-empty support $\|f\|$ of a semiflow f is *minimal* with respect to set inclusion if and only if $\nexists\, g \in \mathcal{F}^+ \setminus \{0\}$ such that $\|g\| \subset \|f\|$.

Since P is finite, the set $\mathcal{MS}$ of minimal supports in P is a *Sperner family* (i.e., a family of subsets such that none of them contains another one) and we can apply Sperner's theorem [Spe28] over $c = |\mathcal{MS}|$ which states that:

$$c \leq \binom{d}{\lfloor d/2 \rfloor}. \tag{11}$$

This result, which was already mentioned in [Mem83, STC98], provides us with a general upper bound for the number of minimal supports in a given Petri Net. To the best of our knowledge, this bound can be reached only by three families of degenerate Petri Nets containing isolated elements: only one, two, or three places and as many isolated transitions as desired. We conjecture that this bound cannot be reached for Petri Nets with more than three places and should be refined on a case-by-case basis by exploiting connectivity between places and transitions as hinted in the example Section 7.3.

A non-null semiflow f is *minimal* with respect to $\leq$ if and only if $\nexists\, g \in \mathcal{F}^+ \setminus \{0, f\}$ such that $g \leq f$.

In other words, a minimal semiflow cannot be decomposed as the sum of another semiflow and a non-null non-negative vector. This remark yields initial insight into the foundational role of minimality regarding the decomposition of semiflows. We are looking for characterizing generating sets such that they allow analyzing various behavioral properties as efficiently as possible.

First, if we consider a generating set over $\mathbb{N}$, then we may have to explore every minimal semiflow. Although finite, the number of minimal semiflows can be quite large. Second, considering a basis over $\mathbb{Q}$ may not capture behavioral constraints quite easily (see the example of Section 7.3).

6.4. Three decomposition theorems

Generating sets can be characterized thanks to a set of three decomposition theorems that can be found in [Mem78] with their proofs. Here, theorem 2 is extended to better characterize minimal semiflows and generating sets over $\mathbb{N}$, and is provided with a new proof using Gordan's lemma 4. Theorem 3 was only valid over $\mathbb{N}$ and is now extended to include $\mathbb{Q}^+$ and $\mathbb{Q}$. Theorem 4 is only valid over $\mathbb{Q}^+$ and is unchanged.

6.4.1. Decomposition over non-negative integers

Theorem 2. *If a semiflow is minimal then it belongs to any generating set over $\mathbb{N}$.*

The set of minimal semiflows of $\mathcal{F}^+$ is a finite generating set over $\mathbb{N}$.

Let's consider a semiflow $f \in \mathcal{F}^+ \setminus \{0\}$ such that $\exists f_1, ..., f_k \in \mathcal{F}^+ \setminus \{0, f\}$ and $a_1, ..., a_k \in \mathbb{N}$ such that $f = \sum_{i=1}^{i=k} a_i f_i$. Since $f \neq 0$ and all coefficients a_i are in $\mathbb{N}$, $\exists j \leq k$ such that $a_j > 0$. Therefore, $a_j f_j \leq f$, since $f_j \neq f$, we have $f_j < f$ so f is not minimal. Hence, if a semiflow is minimal then it has to belong to every generating set over $\mathbb{N}$.

Applying Gordan's lemma, there exists $\mathcal{G}$, a finite generating set. Since any minimal semiflow is in $\mathcal{G}$, the subset of all minimal semiflows is included in $\mathcal{G}$ and therefore finite. Let $\mathcal{E} = \{e_1, ...e_n\}$ be this subset.

For any semiflow $f \in \mathcal{F}^+$, we can always define $r \in \mathcal{F}^+$ and a set of n non-negative integers $\{k_1, ...k_n\}$ such that:

i) $r = f - \sum_{j=1}^{j=n} k_j e_j$, where k_i are defined by:

ii) $\forall i \leq n, (f - \sum_{j=1}^{j=i} k_j e_j) \in \mathcal{F}^+$ and $(f - \sum_{j=1}^{j=i} k_j e_j - e_i) \notin \mathcal{F}^+$

By construction of the non-negative integers k_i , we have $r \in \mathcal{F}^+ \setminus \{0\}$ and $\nexists e_i \in \mathcal{E}$ such that $e_i \leq r$. This means that r is minimal or null. $\mathcal{E}$ includes all minimal semiflows, therefore, if r were minimal, then $\exists i \leq n$ such that $e_i = r$. Again, this would contradict the way the coefficients k_i were defined. Therefore, $r = 0$, any semiflow can be decomposed as a linear combinations of minimal semiflows, and $\mathcal{E}$ is a generating set. [23] $\square$

[23]If $\mathcal{E}$ were infinite, the construction could still be used since the decreasing sequence is bounded by 0 and $\mathbb{N}$ is nowhere dense, and we would have:
$\lim_{n\to\infty} f - \sum_{j=1}^{j=n} k_j e_j = 0$ with the same definition of the coefficients k_j as in ii).

Let's point out that since $\mathcal{E}$ is not necessarily a basis, the exhibited decomposition may not be unique and depends on the order in which the minimal semiflows are considered as shown in Figure 6 where the semiflow $h^\top = (9, 9, 6, 0, 3)$ can be decomposed in three different ways: $h = f_2 + g_1 + f_1 = 3f_1 = 2g1 + g3$ just by changing the order in which the 5 minimal semiflows are considered.

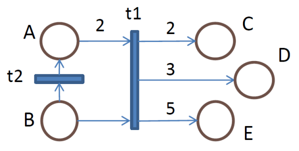


Figure 6: $f_1^\top = (3, 3, 2, 0, 1)$, $f_2^\top = (4, 4, 1, 0, 2)$, $g_1^\top = (2, 2, 3, 0, 0)$, $g_2^\top = (1, 1, 0, 1, 0)$, $g_3^\top = (5, 5, 0, 0, 3)$ are five canonical and minimal semiflows. $\|f_1\|$ or $\|f_2\|$ are not minimal. f_1 and f_2 are linear combinations of g_1, g_2, g_3 over $\mathbb{Q}^+$: $f_1 = \frac{1}{3}(2g_1 + g_3)$ and $f_2 = \frac{1}{3}(g_1 + 2g_3)$. Therefore, the decomposition of a semiflow on $\{f_1, f_2, g_1, g_2, g_3\}$ is not unique. Moreover, $\mathcal{G}_1 = \{g_1, g_2, g_3\}$ is a generating set over $\mathbb{Q}^+$ or over $\mathbb{Q}$.

However, a minimal semiflow does not necessarily belong to a generating set over $\mathbb{Q}^+$ or $\mathbb{Q}$. In Figure 6, $\mathcal{G}_1$ does not include f_1, which is minimal or in Figure 7 where $\mathcal{G}_3$ does not include g_4 which is of minimal support.

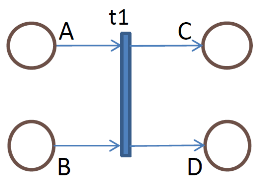


Figure 7: $f^\top = (1, 1, 1, 1)$, $g_1^\top = (0, 1, 1, 0)$, $g_2^\top = (0, 1, 0, 1)$, $g_3^\top = (1, 0, 1, 0)$, $g_4^\top = (1, 0, 0, 1)$ are five canonical semiflows. f is not minimal and $\|f\|$ is not minimal. $\mathcal{G}_2 = \{g_1, g_2, g_3, g_4\}$ is the unique generating set over $\mathbb{N}$ and $f = g_1 + g_4 = g_2 + g_3$ has exactly two different decompositions in $\mathcal{G}_2$. $\mathcal{G}_3 = \{g_1, g_2, g_3\}$ is a generating set over $\mathbb{Q}$.

6.4.2. Decomposition over semiflows of minimal support

Theorem 3. *If I is a minimal support then*

i) there exists a unique minimal semiflow f such that $I = \|f\|$ and $\forall g \in \mathcal{F}^+$ such that $\|g\| = I, \exists k \in \mathbb{N}$ such that $g = kf$, *ii) any non-null semiflow g such that $\|g\| = I$ constitutes a generating set over $\mathbb{Q}^+$ or $\mathbb{Q}$ for $\mathcal{F}_I^+ = \{g \in \mathcal{F}^+ | \ \|g\| = I\}$.*

In other words, $\{f\}$ is a unique generating set over $\mathbb{N}$ for $\mathcal{F}_I^+ = \{g \in \mathcal{F}^+ \mid \|g\| = I\}$. However, this uniqueness property is indeed lost in $\mathbb{Q}^+$ or in $\mathbb{Q}$, since any element of $\mathcal{F}_I^+$ is a generating set of $\mathcal{F}_I^+$ over $\mathbb{Q}^+$ or $\mathbb{Q}$.

From Sperner's theorem, any support I of a semiflow contains a finite number m of minimal supports of semiflows. The following theorem states that these m supports cover I, and provide a generating set deduced from these m supports.

Theorem 4. *(decomposition) Any support I of semiflows is covered by the finite subset $\{I_1, I_2, \ldots, I_m\}$ of minimal supports of semiflows included in I:*
$I = \bigcup_{i=1}^{i=m} I_i$.

Moreover, $\forall f \in \mathcal{F}^+$ such that $\|f\| = I$, one has $f = \sum_{i=1}^{i=m} \alpha_i g_i$ where $\alpha_i \in \mathbb{Q}^+$ and the semiflows g_i are such that $\|g_i\| = I_i$.

A sketch of proof of theorem 4 can be found in [Bra82], a complete proof in [Mem78].

6.5. Canonical semiflows

A semiflow is *canonical* ([CS89], [CST03] p. 68) if and only if the gcd of its non-null coordinates is equal to one. In [CMPAW09], such a semiflow is said to be *scaled back*.

6.5.1. Canonical and minimal semiflows

Minimal semiflows and canonical semiflows are two different notions. The following lemma and theorem help to compare them.

Lemma 5. *If a semiflow is minimal then it is canonical.*

If a semiflow is canonical and its support is minimal then it is minimal.

The first point is quite evident: if f is not canonical its gcd k is such that $k > 1$ so $\exists g \in \mathcal{F}^+$ such that $f = kg$ and f would not be minimal.

The second point is a direct application of theorem 3. $\square$

However, canonical semiflows are not necessarily minimal semiflows; minimal semiflows do not necessarily have a minimal support. For example, Figure 6, f_1 and f_2 are canonical and minimal, but their support is not minimal. Any semiflow $f = ag_1 + g_2 + bg_3$ where $a, b \in \mathbb{N}$ and $a + b > 0$ is canonical and clearly not minimal. As a and b can be arbitrarily large, this shows that the number of canonical semiflows can be infinite.

6.5.2. About the number of canonical semiflows

We can observe in the previous example that the infinite sequence of canonical semiflows is constituted of semiflows with non-minimal support. This is hinted on the following theorem.

Theorem 5. *Given a support I, $c_I = |\{f \text{ canonical semiflow} \mid \|f\| = I\}|$,*

If I is a minimal support, then $c_I = 1$, else c_I is infinite.

If $c_I = 1$, then I is minimal.

From theorem 3, there is a unique minimal semiflow having a given minimal support. From lemma 5, a minimal semiflow is canonical. Hence, if I is minimal, then $c_I = 1$.

If I is not minimal, then $\exists\ e,\ f \in \mathcal{F}^+$ such that $\|e\| \subset \|f\| = I$. We can build an infinite sequence of semiflows $f_i,\ i \in \mathbb{N}$, such that $f_i = \alpha_i(f + k_i e)$, where $k_i \in \mathbb{N}$ and $1/\alpha_i$ is the gcd of the non-null coordinates of $f + k_i e$. $\forall i \in \mathbb{N}. f_i$ is canonical by construction. Let's consider i, j such that $f_i = f_j$; then $\alpha_i(f + k_i e) = \alpha_j(f + k_j e)$. This leads to: $(\alpha_i - \alpha_j)f = (k_i - k_j)e$. However, since $\|e\| \subset \|f\|$, we must have $\alpha_i = \alpha_j$ and $k_i = k_j$. Hence, we built an infinite sequence of canonical semiflows based upon an infinite sequence of non-negative integers.

If $c_I = 1$ then let f be the unique canonical semiflow of support I. Let's consider $g \in \mathcal{F}^+$ such that $\|g\| \subseteq I$. With the same construction as before, we can build a canonical semiflow $h = \alpha(f + kg)$ where $1/\alpha$ is the gcd of the non-null coordinates of $f + kg$, and $k \in \mathbb{N}$. We have $\|h\| = I$ and $c_I = 1$; therefore, $h = \alpha(f + kg) = f$. Then, $g = ((1 - \alpha)/k\alpha)f$ which means that any semiflow of support included in I is a multiple of f. Hence, I is minimal. $\square$

The fact that the number of canonical semiflows can be infinite was already pointed out in [KJ87, CS89, CMPAW09]. The fact that this number is infinite only when the considered support is non minimal as described in theorem 5 is new to the best of our knowledge.

6.6. Minimal generating sets, least generating sets, and fundamental sets

Minimal generating sets have been defined over $\mathbb{N}$ in [Mem78] and over $\mathbb{Q}^+$ in [Mem78, Mem83], and least generating sets over $\mathbb{Q}$ in [CS89, GV03]. Similarly to the notion of generating set defined in Section 6.2, we slightly extend their definition to hold over a set $\mathbb{S} \in \{\mathbb{N},\ \mathbb{Q}^+, \mathbb{Q}\}$.

From [Mem83] p. 39, a *minimal generating set* over $\mathbb{S}$ is a generating set that does not strictly include any generating set.

From [CS89] p. 82, or [CST03] p. 68, a *least generating set of semiflows* "is made up of the least number of elements to generate any semiflow" over $\mathbb{S}$ [24] In other words, $\mathcal{G}$ is a least generating set if and only if it does not exist a generating set $\mathcal{H}$ such that $|\mathcal{H}| \leq |\mathcal{G}|$.

6.6.1. Coincidence between minimal and least generating sets

A minimal generating set is defined with respect to set inclusion while a least generating set is defined with respect to its cardinality. In the case of generating sets of semiflows, theorem 6 hereunder is a new result stating that these two different notions are in fact equivalent over $\mathbb{S}$.

Lemma 6. *If $\mathcal{G}$ is a generating set over $\mathbb{Q}^+$ or $\mathbb{N}$, I a minimal support, then $\exists g \in \mathcal{G}$ such that $I = \|g\|$.*

We consider e a semiflow of minimal support, $\mathcal{G} = \{g_1, ...g_k\}$, a generating set over $\mathbb{Q}^+$ or $\mathbb{N}$. Then, $e = \sum_{i=1}^{i=k} \alpha_i g_i$. All the coefficients are non-negative and $e \neq 0$, then $\exists j \leq k$ such that $\alpha_j > 0$ and $e \geq \alpha_j g_j$. Since $\|e\|$ is minimal, $\|e\| = \|g_j\|$. $\square$

[24] More precisely, the least generating set is defined over $\mathbb{Q}$ in [CS89, CST03] and over $\mathbb{N}$ in [CMPAW09].

This lemma states that any generating set over $\mathbb{Q}^+$ or $\mathbb{N}$ contains at least one semiflow per minimal support. Indeed, this property is not true over $\mathbb{Q}$. In Figure 7, $\mathcal{G}_2$ is a minimal generating set over $\mathbb{Q}^+$ and $\{g_2, g_3, g_4\} \subset \mathcal{G}_2$ is a generating set over $\mathbb{Q}$ since $g_1 = g_2 + g_3 - g_4$ is of minimal support but generated over $\mathbb{Q}$ (since one coefficient is negative) by the other minimal semiflows of minimal support.

Theorem 6. *If $\mathcal{G}$ is a generating set over $\mathbb{S}$, where $\mathbb{S} \in \{\mathbb{N}, \mathbb{Q}^+, \mathbb{Q}\}$ then the two following properties are equivalent:*

$\mathcal{G}$ is a minimal generating set,

$\mathcal{G}$ is a least generating set.

First, the fact that a least generating set is a minimal generating set is straightforward.

Let's consider $\mathcal{G}$, a minimal generating set over $\mathbb{N}$. By applying theorem 2, we conclude that $\mathcal{G}$ is the set of minimal semiflows and a least generating set.

Next, let's consider $\mathcal{G}$, a minimal generating set over $\mathbb{Q}^+$. Then, lemma 6 can apply stating that $\mathcal{G}$ includes $\mathcal{G}'$ a family of exactly one semiflow for each minimal support. From theorem 4, we draw that $\mathcal{G}'$ is a generating set. $\mathcal{G}$ is minimal then $\mathcal{G} = \mathcal{G}'$. This being true for any minimal generating set, $\mathcal{G}$ is also a least generating set.

Finally, let's consider $\mathcal{G}$ a generating set over $\mathbb{Q}$. From $\mathcal{G}$ we can extract a subset $\mathcal{B}$ of linearly independent semiflows (see [Lan02] p. 85 for basic results on vector spaces). Then, $\mathcal{B}$ is a least and minimal generating set over $\mathbb{Q}$. $\square$

6.6.2. About fundamental sets

Theorem 5 states that there is exactly one canonical semiflow for each minimal support. This particularity characterizes the notion of a fundamental set.

In [STC98] p. 319, the set of all canonical semiflows of minimal support is called a *fundamental set.*

Corollary 2. *(fundamental set) A fundamental set is a generating set over $\mathbb{Q}^+$ or $\mathbb{Q}$ but not necessarily over $\mathbb{N}$. A fundamental set over $\mathbb{Q}^+$ is a minimal* *generating set but not necessarily over* $\mathbb{Q}$.

The first point of this corollary is a direct consequence of theorem 4 and lemma 5: we conclude that a fundamental set is one possible generating set over $\mathbb{Q}^+$ and therefore over $\mathbb{Q}$.

The second point is directly deduced from the first point and lemma 6.

The last parts of the two points of the corollary are illustrated by the two following counterexamples.

In Figure 6, $\mathcal{G}_1 = \{g_1, g_2, g_3\}$ is a fundamental set that is not a generating set over $\mathbb{N}$ since f_1 or f_2 are minimal and cannot be decomposed as a linear combination of elements of $\mathcal{G}_1$ over $\mathbb{N}$ [25].

A fundamental set over $\mathbb{Q}$ is not necessarily a minimal generating set; in Figure 7, $\mathcal{G}_2$ is the fundamental set and is not a minimal generating set. $\square$

Belonging to a least generating set, hereunder denoted by *lgs*, does not equip a semiflow f with specific properties [26]:

- if $f \in lgs$ over $\mathbb{N}$, then f is minimal but not necessarily canonical or of minimal support;

- if $f \in lgs$ over $\mathbb{Q}^+$, then f has a minimal support but is not necessarily canonical or minimal;

- if $f \in lgs$ over $\mathbb{Q}$, then f is not necessarily minimal and not necessarily canonical or of minimal support.

6.6.3. About uniqueness

Corollary 3. *(uniqueness) The set of minimal semiflows is the unique minimal generating set over* $\mathbb{N}$.

If $\mathcal{G}$ *is a least generating set over* $\mathbb{Q}^+$ *then for any minimal support* I *of semiflows,* $\exists g \in \mathcal{G}$ *unique such that* $I = \|g\|$.

[25] In this regard, the statement p.143-147 of [CMPAW09] should be rewritten.

[26] The properties 2.2 p. 82 of [CS89] and 5.2.5 p.68 of [CTSH03] should be rewritten by taking the following statements into account.

The fundamental set is the unique minimal generating set of minimal semi-flows over $\mathbb{Q}^+$.

The first point can be directly deduced from theorem 2, the second and third are directly deduced from theorems 3 and 4 and lemma 5. □

The third point of this corollary can be considered as a variation of a statement in [STC98].

However, a minimal generating set over $\mathbb{Q}^+$ or $\mathbb{Q}$ is not unique even among minimal semiflows of minimal support. In the example of Figure 7,

$\{k_1 g_1, k_2 g_2, k_3 g_3, k_4 g_4\}$ where $k_i \in \mathbb{N}$ constitutes a family of minimal generating sets over $\mathbb{Q}^+$. Moreover, $\mathcal{G}_3 = \{g_1, g_2, g_3\}$ and $\{g_2, g_3, g_4\}$ are two minimal generating sets over $\mathbb{Q}$.

6.7. Results summary

We have seen through several counterexamples that no result must be taken for granted and that any proposition must be carefully addressed. Figure 8 summarizes the main results presented in this note.

Our contribution in terms of new results consists of the following:

- the first three theorems in Section 6.4, which have been slightly extended with a new proof for theorem 2 and could be used for infinite Petri Nets [Pet96];

- lemma 5 and theorem 5 of Section 6.5 about comparing minimal and canonical semiflows;

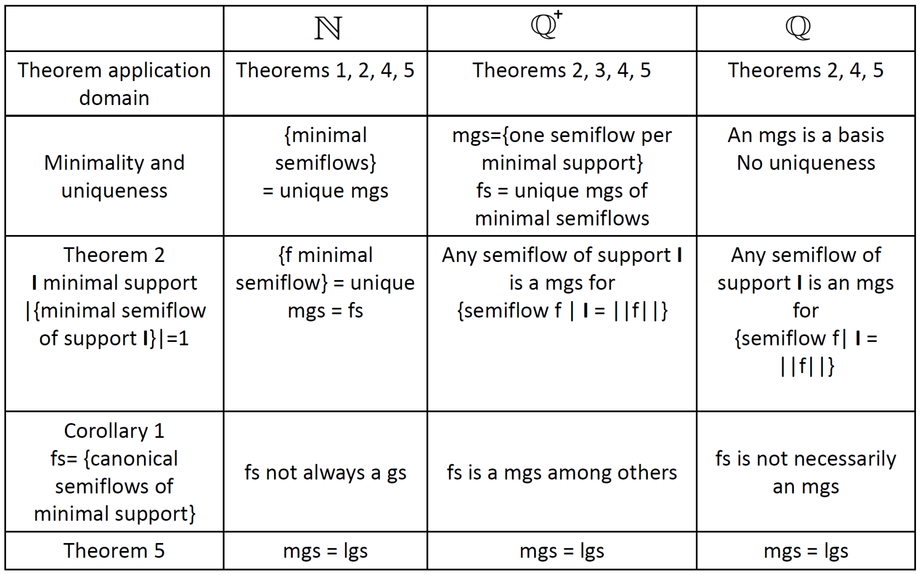
- lemma 6 and theorem 6 about the equivalence between minimal and least generating sets;

- corollaries 2 and 3 about fundamental sets and uniqueness.

Let us illustrate some of the above concepts through a few examples.

7. Reasoning with invariants and home spaces

Invariants can be used to prove a rich array of behavioral properties of conceptual models such as LTS or Petri Net even within different settings. Analysis can be performed with incomplete information on the initial marking as shown



	$\mathbb{N}$	$\mathbb{Q}^+$	$\mathbb{Q}$
Theorem application domain	Theorems 1, 2, 4, 5	Theorems 2, 3, 4, 5	Theorems 2, 4, 5
Minimality and uniqueness	{minimal semiflows} = unique mgs	mgs={one semiflow per minimal support} fs = unique mgs of minimal semiflows	An mgs is a basis No uniqueness
Theorem 2 **I** minimal support |{minimal semiflow of support **I**}|=1	{f minimal semiflow} = unique mgs = fs	Any semiflow of support **I** is a mgs for {semiflow f | **I** = ||f||}	Any semiflow of support **I** is an mgs for {semiflow f| **I** = ||f||}
Corollary 1 fs= {canonical semiflows of minimal support}	fs not always a gs	fs is a mgs among others	fs is not necessarily an mgs
Theorem 5	mgs = lgs	mgs = lgs	mgs = lgs

Figure 8: gs, fs, mgs, lgs denote the generating set, the fundamental set, the minimal generating set, and the least generating set respectively.

in the first example below or on a subsystem exhibiting some compositionality ability. It can be described with parameters which will make the invariant calculus more complex but still tractable as shown in the subsequent example. It is then, sometimes though not always, possible to conclude avoiding a painstaking symbolic model checking or a parameterized and complex development of a reachability graph [DRvB01].

7.1. A tiny example

The Petri Net in Figure 9 [27] is defined by:

$Pre(\cdot, t_1)^\top = (2, 0); Pre(\cdot, t_2)^\top = (1, 1);$

$Post(\cdot, t_1)^\top = (0, 1); Post(\cdot, t_2)^\top = (3, 0).$

$f^\top = (1, 2)$ is a semiflow: the Kirchhoff's law is easily verified for the nodes t_1 and t_2. Moreover, it is obvious that f is a minimal semiflow and its support

[27] This example can be first found in [Bra82] or in [Mem83] without proof then in [Mem19] with the main elements of the proof that is completed here.

is minimal as well.

It models a formal machine semi-deciding whether a given natural number n is even or null. Here, we just mean that it will always be possible to find a sequence of transitions such that the Petri Net stops (t_1 and t_2 are not enabled anymore, i.e. there exists a sink) given $q_0(A) = 2k$ where $k \in \mathbb{N}$, regardless of $q_0(B)$, the initial marking of B. Actually, we can do slightly better and also prove that this Petri Net is live if and only if $f^\top q_0 = 2i + 1$ where $i \in \mathbb{N}^+$.

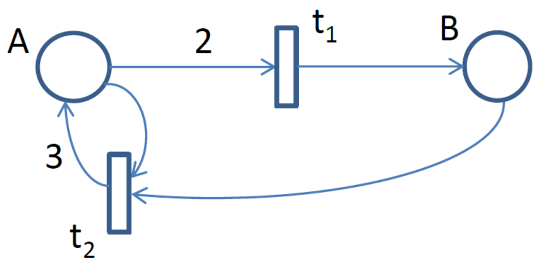


Figure 9: This tiny Petri Net is live if and only if $f^\top q_0$ is an odd number greater than one whatever is the initial marking of B.

The scalar product $f^\top q$ does not vary, we have the following invariant: for any marking q reachable from an initial marking q_0, $f^\top q = q(A)+2q(B) = f^\top q_0$. This can be rewritten as $q(A) = f^\top q_0 - 2q(B)$ which means that $q(A)$ and $f^\top q_0$ have always the same parity and this will never change [28] (if we admit that zero is even). Therefore, if $f^\top q_0 > 2$ and is odd, then $q(A)$ remains always odd. If $q(A) = 1$ and $f^\top q_0 > 2$ then $q(B) > 1$ and t_2 is enable from q; it can be deduced that the Petri Net is live since $q(A)$ remains odd; if $q(A) > 1$ then t_1 is enable from q; again, the liveness of the Petri Net can be easily deduced. Reversely, if the Petri Net is live then $f^\top q_0 > 2$ since the enabling threshold of t_2 is $f^\top Pre(\cdot, t_2) = 3$. Moreover, if $f^\top q_0$ is even, then $q(A)$ is even and the Petri Net is not live (t_1 can be enabled until A has no token which means that

[28]Let's point out that we just used the phrases *always* and *never will* as mentioned in Section 4.1: we expressed an invariant about the parity of the marking of the place A. This is not the usual formulation for an invariant generated by a semiflow.

it is not possible to enable t_1 or t_2 anymore) $\square$

Since $f > 0$, we have A and B bounded by $f^\top q_0$ for any initial marking q_0. $\{A, B\}$ is therefore structurally bounded.

What is remarkable about the analysis of this tiny example is that it was not necessary to develop the reachability graph in order to decide whether or not the Petri Net is live or bounded. We could analyze the Petri Net even partially ignoring the initial marking (i.e. considering $q_0(A)$ as a parameter and without even considering the value taken by $q_0(B)$).

An interesting behavioral lesson to draw from this tiny example has to do with liveness: once live, adding more tokens to some places of the Petri Net does not guarantee that liveness will persist. As a matter of fact, while a minimum of tokens are required (refer for instance to the notion of enabling threshold in 5.3.2) for a PN to be live, in this example, adding one token to A will make the Petri Net not live while adding a second token will make it live again.

Even if adding more resources to the system under study can bring confidence that some actions will eventually be performed, this does not necessarily result in solving a deadlock issue; this could even create a deadlock situation! Adding more resources is a false "good idea" that is encountered with many students and engineers certainly taking its origin with the notion of monotonicity seen section 3.9.

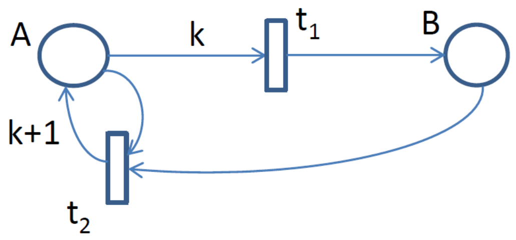


Figure 10: This second tiny Petri Net is live if and only if $g^\top q_0 > k$ and is not a multiple of k whatever is the initial marking of B. For $k = 1$, the Petri Net has no live transition whatever is the initial marking. For $k = 2$, we find again the Petri Net of Figure 9.

We can expand the tiny example of Figure 9 by adding a parameter k such that $k > 1$ as in Figure 10 and call TNK this Petri Net. This time, we have the semiflow: $g^\top = (1, k)$ and we can show that the Petri Net is not live if and only if $g^\top q_0 \leq k$ or $g^\top q_0 = nk$ where $n \in \mathbb{N}$, independently of $q_0(B)$.

If $g^\top q_0 < k$ then the enabling threshold of t_1 can never be reached and neither t_1 nor t_2 can be executed. The Euclidean division of $g^\top q_0$ by k gives: $g^\top q_0 = nk + i$ where $i < k$ then, $g^\top q \equiv i \ (mod\ k)$ therefore $q(A) \equiv i \ \forall q \in RS(TNK, q_0)$. If $i = 0$ then we have $q(A) = nk - kq(B)$ and t_1 can be enabled $n - q(B)$ times to reach a marking with zero token in A. If $i \neq 0$ and $g^\top q_0 > k$ then $q(A) \neq 0$ and either $q(A) > k$ or $q(B) \neq 0 \ \forall q \in RS(TNK, q_0)$. In the first case, t_1 is enabled in the second case t_2 is enabled. It is easy to conclude that the Petri Net TNK is live. $\square$

The loop between A and t_2 prevents to draw this kind inof result with the sole usage of the state equation (4).

Curiously, we conjecture that to design another Petri Net semi- deciding whether a given number is odd is impossible.

7.2. Revisiting the mutual exclusion example

One could argue that the Petri Nets of Figures 5, 9, or 10 are relatively simple and that it would be easy to obtain similar results without reasoning with semiflows. This may be true (for instance, using the LRG of the Petri Net of Figure 5 is particularly trivial), however, it must be stressed out that semiflows allow to elegantly develop reasoning in the presence of parameters.

In this regard, let's revisit and generalize our mutual exclusion example of Section 4.2.3 by introducing few parameters. This time, we allow k instances of Program 1 and l instances of Program 2 to run concurrently. The two edges from $\mathbf{S}$ to $Semp_1$ and from $Semv_1$ to $\mathbf{S}$ are labeled by x; the two edges from $\mathbf{S}$ to $Semp_2$ and from $Semv_2$ to $\mathbf{S}$ are labeled by y (by construction, we have $x > 0$ and $y > 0$). The initial state q_0 is being defined with three non-negative parameters: k tokens in A, l tokens in D, and z tokens in $\mathbf{S}$; and, $q_0(B) = q_0(E) = 0$.

In this example, we authorize several instances of the same program to be simultaneously in their `critical section` however, we still want to verify mutual exclusion between instances of Program 1 and instances of Program 2 that is to say that we must have either $q(B) > 0$ or $q(E) > 0$ for any reachable marking. From the four versions of invariant representing mutual exclusion in section 4.2.3 only (ME) and (ME)' still make sense in this parameterized context. Therefore, we want to determine under which conditions (i.e. which values of the parameters) the Petri Net is live and:

(ME): $q(B) \times q(E) = 0$ for any marking reachable from q_0 is an invariant.

The Petri Net Figure 11 has still 3 invariants directly deduced from 3 semiflows. The semiflows f_1 and f_2 are unchanged from Figure 5; the third semiflow sem_2 is different and such that: $sem_2(B) = x$, $sem_2(E) = y$, $sem_2(\mathbf{S}) = 1$, $sem_2(p) = 0$ for any other place p. For any reachable marking q from q_0,we have:

$I : x \times q(B) + y \times q(E) + q(\mathbf{S}) = z$

$F_1 : q(B) + q(A) = k$

$F_2 : q(D) + q(E) = l$

We propose to mead this proof in three steps.

First, it is easy to prove that q_0 is a home state: let's assume $u \in \mathbb{N}$, $v \in \mathbb{N}$ and $q \in RS$ such that $q(B) = u$ and $q(E) = v$. Therefore, we can enable the sequence $Semv_1^u Semv_2^v$ for any value of u and v. We then reach a marking q' such that $q'(B) = q'(E) = 0$. From I, F_1, F_2, we easily conclude that $q' = q_0$ without any assumption on the values of the parameters k, l, z, x, y.

Second, let us see that the Petri Net is live if and only if:

$k > 0, l > 0$ and $z \geq max(x, y)$.

If the Petri Net is live then F_1 (saying that there is no more than k tokens in A) involves $k = q_0(A) > 0$, F_2 involves $l = q_0(D) > 0$ and I involves that $q(\mathbf{S})$ is bounded by z, therefore we must have $z \geq max(x, y)$ to have a chance to reach the enabling threshold of $Semp_1$ or $Semp_2$ and enable them. Reversely, if the 3 inequalities hold, we use the fact that q_0 is a home state to easily deduce that $Semp_i$ are live then $Semv_i$ are also live (since they need only B or E to contain one token that $Semp_i$ would have produced) and the Petri Net is live.

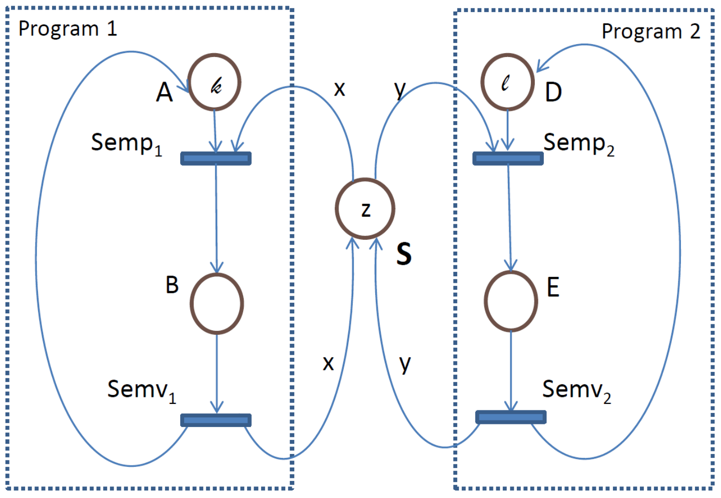


Figure 11: k instances of Program1 and l instances of Program 2 can be executed concurrently. This time, unlike the Petri Net in Figure 5, the mutual exclusion invariant depends on the various values of the three parameters x, y, z independently of the parameters k and l. This cannot be straightforwardly deduced from any semiflow.

Third, we can now prove that (ME) holds if and only if $z < x + y$ or $k = 0$ or $l = 0$. The condition is obviously necessary otherwise if $(z \geq x+y)$ *and* $(k \neq 0)$ *and* $(l \neq 0)$ then we could enable the sequence $Semp_1 Semp_2$ from q_0 and (ME) does not hold. Reversely, if $k = 0$ or $l = 0$ (ME) is obviously verified since one of the two program would have no instance. Now, from $z < x + y$ and I we can write:

$q(\mathbf{S}) = z - x \times q(B) - y \times q(E) < x + y - x \times q(B) - y \times q(E)$

If $q(B)$ and $q(E)$ were both strictly positive then we would have: $q(\mathbf{S}) < 0$ which is impossible. Therefore, $q(B)$ and $q(E)$ cannot be both strictly positive for the same marking and the invariant is satisfied independently of the values of k and l.

We can conclude that this PN coupled with q_0 is live and (ME) is an invariant if and only if $k > 0$, $l > 0$, and $z \in [max(x, y), x + y - 1]$. $\square$

Moreover, let us assume $z < x+y$ and $x < y$. We have just seen that: $q(\mathbf{S}) < x \times (1-q(B)) + y \times (1-q(E))$. If $x < y$ then $0 \leq q(\mathbf{S}) < -x \times q(B) + y \times (2-q(E))$

which can hold ony if $q(E) < 2$ which means that: (ME2): if $x < y$ and $z < x+y$ then there cannot be more than one token in E for any reachable marking q from q_0 is a second invariant for this Petri Net.

We can prove similarly that if $x = y$ and $z < x+y$ then we have: $0 < x \times (2 - q(E) - q(B))$. therefore, (ME3): if $x = y$ and $z < x + y$ then $q(B) + q(E) \leq 1$ for any reachable marking q from q_0 is an invariant.

7.3. An example from the telecommunication industry

The example described in Figure 12 is a reduced [29] version of a Petri Net published in [MM81] representing two subscribers, “a caller” and a “callee,” having a conversation (places CLA and CA respectively). Initially, they are in an idle state with places LA and A marked with one token. Signals PU, R are sent from the caller to the callee and signals S, F from the callee to the caller. The overall desired behavior is that caller and callee cannot go back to their idle state as long they have not received all the signals sent to them despite the fact that they both can hang up at any time making the order in which signals F and R are sent and received undetermined.

From their idle state (place LA), the caller can pick up their phone (transitions t_1) sending the signal PU to the callee. From their idle state (place A), the callee, upon receiving the signal PU, can pick up their phone (transition t_7), send the signal S and go to conversation (place CA) from where they can hang up (transition t_8) at any time sending the signal F to the caller. Receiving the signal S, the caller can go (transition t_2) to the conversation (place CLA). They can also hang up at any time (transitions t_3, t_4, t_5) sending the signal R to the callee. After hanging up via t_4 or t_5, the caller will have to wait (place W) until they receive the signal F from the callee before going back (transition t_6) to their initial idle state LA. The callee can go back to their idle state A only upon receiving the signal R (transition t_9).

[29]See [Bra82, GV03] for Petri Nets reduction rules.

The initial marking M_0 is such that $M_0(LA) = M_0(A) = 1,\ M_0(p) = 0$ for any other place: the modeled system is in its idle state.

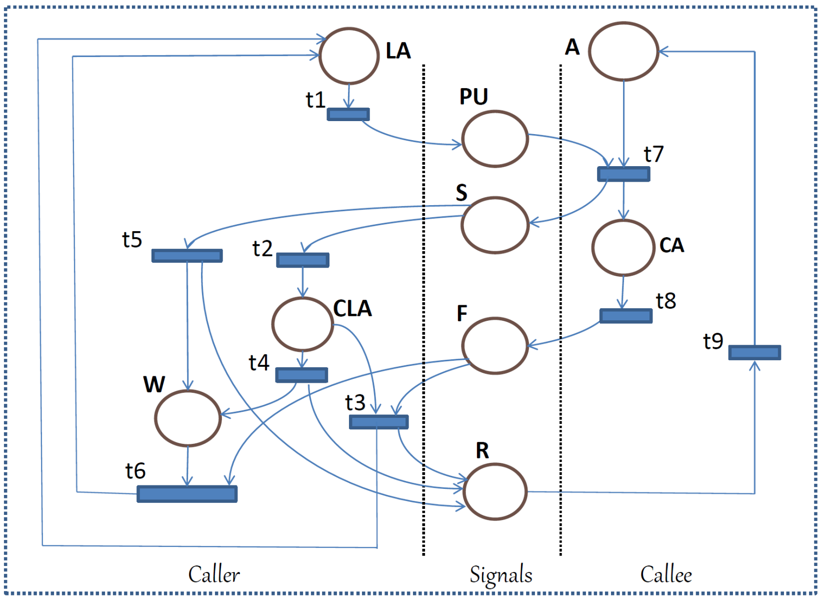


Figure 12: This Petri Net has exactly three minimal semiflows of minimal support constituting a minimal generating set over $\mathbb{Q}^+$: $\mathcal{GB}_1 = \{f_1,\ f_2,\ f_3\}$ such that:
$f_1(LA) = f_1(CLA) = f_1(W) = f_1(PU) = f_1(S) =\ 1,\ f_1(p) =\ 0$ for any other place,
$f_2(LA) = f_2(PU) = f_2(F) = f_2(CA) =\ 1,\ f_2(p) =\ 0$ for any other place,
$f_3(CLA) = f_3(S) = f_3(R) = f_3(A) =\ 1,\ f_3(p) =\ 0$ for any other place.

7.3.1. Optimizing Sperner's bound

First, let us notice that inequality (11) gives us the following bound for m, the number of minimal supports: $m \leq \binom{9}{\lfloor 9/2 \rfloor} = 126$. However, it is easy to notice that transitions $t_1, t_2, t_8,$ and t_9 have only one input and one output, which means that any support including such an input also includes the corresponding output to satisfy the equations associated to $t_1, t_2, t_8,$ and t_9. The bound can be optimized to: $m \leq \binom{5}{\lfloor 5/2 \rfloor} = 10$. It can further be improved by reasoning on transitions t_5 and t_6 and reach $m \leq 3$.

7.3.2. A proof scheme using minimal semiflows of minimal support

We want to verify the following property (known as safeness [GV03] p.489): $\mathcal{P} = \forall M$ reachable from $M_0, \forall p \in P, M(p) \leq 1$.

We also want to prove that the initial marking M_0 is a home state (i.e., a marking such that whatever the evolution of the Petri Net, it is always possible to reach it back) from which it is easy to deduce that the Petri Net is live.

These two important properties are proven hereunder, starting from $\mathcal{GB}_1 = \{f_1, f_2, f_3\}$ the set of minimal semiflows of minimal support defined Figure 12 without considering every reachable marking.

Similarly as in equation (8), three invariants can directly be drawn from $\mathcal{GB}_1$: for any reachable marking M from M_0, $f_1^\top M = f_1^\top M_0 = 1$, $f_2^\top M = f_2^\top M_0 = 1$, $f_3^\top M = f_3^\top M_0 = 1$. [30]

In other words, there is exactly one token in the support of any semiflow of $\mathcal{GB}_1$. Hence, $\mathcal{P}$ is true since $P = \bigcup_{i=1}^{i=3} \|f_i\|$.

In order to prove that M_0 is a home state, we start to prove that $\mathcal{HS} = \{M \mid M(LA) = 1\}$ is a home space (i.e., a set of markings such that whatever is the evolution of the Petri Net, it is always possible to reach one element of the set). If we consider the invariant deduced from f_2, we know that any reachable marking M is necessarily in one of the 4 following cases.

i) $M(LA) = 1$, then $M \in \mathcal{HS}$,

ii) $M(F) = 1$, since $f_2^\top M = 1$, we have $M(LA) = M(PU) = M(CA) = 0$. $f_1^\top M = f_1(CLA)M(CLA)+f_1(W)M(W)+f_1(S)M(S) = 1$ and it then remains only 3 sub-cases to explore:

ii-1) $M(CLA) = 1$, t_3 can occur from M and we can reach $M' \in \mathcal{HS}$

ii-2) $M(W) = 1$, t_6 can occur from M and we can reach $M' \in \mathcal{HS}$

ii-3) $M(S) = 1$, t_2 can occur from M then we are in the sub-case ii-1).

iii) $M(CA) = 1$, t_8 can occur from M then we are in the case ii).

iv) $M(PU) = 1$, then considering the invariants deduced from f_1 and f_2, we have: $M(LA) = M(CLA) = M(W) = M(F) = M(S) = M(CA) = 0$

[30] In the following, we will omit the phrase “for any reachable marking M from M_0”.

From $f_3^\top M = 1$, we have two sub-cases:

iv-1) $M(A) = 1$, t_7 can occur from M then we are in the case iii).

iv-2) $M(R) = 1$, t_9 can occur then we are in the case iv-1) again.

From these simple 4 cases and 5 sub-cases, we directly deduce that $\mathcal{HS}$ is a home space from where it is easy to conclude that M_0 is a home state and the Petri Net is live.

7.3.3. Further remarks

We let the interested reader develop the same proof scheme from $\mathcal{GB}_2 = \{f_1,\ g = f_2 + f_3,\ h = f_1 + f_3\}$ or from $\mathcal{GB}_3 = \{g,\ h,\ l = f_1 + f_2\}$ which are minimal generating sets over $\mathbb{Q}$ and find out that the analysis becomes more complex even with small additional changes. We conclude that the smaller the support, the more effective the analysis will be since the number of cases and sub cases of the proof scheme depends on the number of elements of each considered support.

The same proof scheme can be used with a colored Petri Net or the same Petri Net enriched with two parameters to model x callers and y callees. In the latter, the initial marking M_0 becomes: $M_0(LA) = x,\ M_0(A) = y,\ M_0(p) = 0$ for any other place p. $\mathcal{GB}_1$ is unchanged, but its three associated invariants become: $f_1^\top M = f_1^\top M_0 = x$, $f_2^\top M = f_2^\top M_0 = x$, $f_3^\top M = f_3^\top M_0 = y$. To prove that $\mathcal{HS} = \{M|\ M(LA) = x\}$ is a home space, we can start from a marking M that satisfies $f_2^\top M = x$ and build a sequence of transitions σ_1 that will reach a marking M' such that $M'(PU) < M(PU)$. σ_1 can then be iterated until a marking of $\mathcal{HS}$ is finally reached. From there, it becomes easy to prove that the Petri Net is live. At last, with the same scheme, it can be proven that $\mathcal{HS}_c = \{M|\ M(CLA) = M(CA) = z \text{ and } z \leq min(x,y)\}$ is a home space meaning that it is always possible to have z subscribers in a conversation.

7.4. A brief illustration of fairness and starvation

The Petri Net Figure 11 has a *fairness* issue despite the fact that it is live for $k > 0$, $l > 0$, and $z \in [max(x,y), x + y - 1]$. As soon as $k > 1$ and $z \geq 2x$, Program1 can *starve* Program2 by having constantly at least one instance of program1 in its `critical section`: the marking of B remains strictly positive and an infinite loop between $Semp_1$ and $Semv_1$ lets no instance of program2 to ever have the opportunity to enter in its `critical section`. Similarly, as soon as $l > 1$ and $z \geq 2y$, Program2 can *starve* Program1: there exists an infinite computation such that $Semp_1$ can never have the opportunity to be enable from any marking of this sequence.

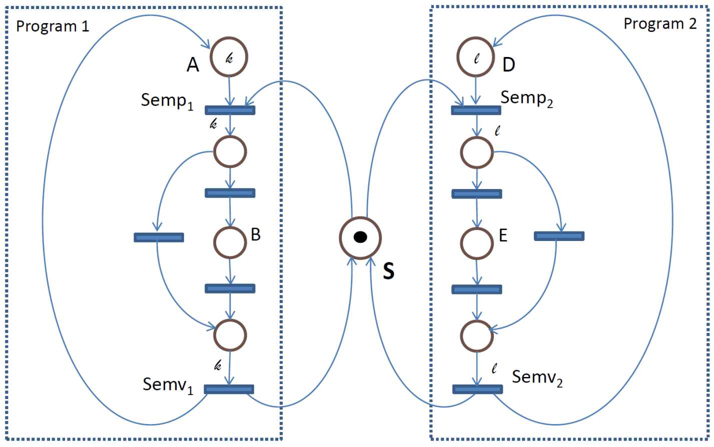


Figure 13: There does not exist an infinite progression such that a Program i can prevent the other one to have the opportunity to execute its `critical section`, in other words, to never have the possibility for $Semp_j$ (where $j = 3 - i$) of being enabled.

The Petri Net Figure 13 addresses this issue since we can prove that the transition $Semv_1$ is enabled only when the marking of B is null. This solution can be compared to another example page 111 of [Bra82] where a privilege is given to one Program to starve the other one. This time, it can be proven that any infinite sequence offers to $Semp_i$ an infinite number of opportunities to be enabled.

8. Conclusion

As soon as we can associate the behavior of a system under study with a set of state variables and follow its evolution through sequences of states forming a reachability graph (or a structure) where each edge is associated with a transition, we can assume the existence of an underlying Transition System. Doing so, invariants and home spaces can be checked or proven along all possible sequences and help understanding and verifying the functioning of such a system.

About Petri Nets, it can never be sufficiently stressed how the combination of their two key concepts puts Petri Nets apart from many other conceptual models. First, a bipartite graph (a topology) captures and models so well the relationship and dependencies between actions and resources. Second, the *"token game"* allows for simulations and construction of a reachability graph given an initial marking (or state). This combination is clearly able to depict concurrency and is supporting somehow a notion of parallelism. It has been seen how semiflows create a link from the static topology (the bipartite graph) to the dynamic evolution (the variation of the number of tokens) of the Petri Net. They support constraints over all possible markings which greatly help analyzing and discovering behavioral properties (even some unspecified ones). Semiflows infer a class of invariants with a constant evaluation function that can be computed from the initial marking of the Petri Net under consideration. Last but not least, they can be characterized by a generating set that can be used in order to support some elegant level of behavioral analysis even with the presence of some level of parameterization.

By considering $\mathbb{N}$, then $\mathbb{Q}^+$, then $\mathbb{Q}$, the size of a minimal generating set decreases as expected since an increasing amount of possibilities to combine semiflows are provided. More interestingly, if m is the number of minimal supports in a Petri Net then:

a minimal generating set over $\mathbb{N}$ is finite and has at least m elements,

a minimal generating set over $\mathbb{Q}^+$ has exactly m elements,

a minimal generating set over $\mathbb{Q}$ has, at most, m elements.

Moreover, we were able to regroup and clarify some key algebraic results scattered in the literature by considering only the two notions of minimality in terms of semiflow and support respectively.

The example of Section 7.3 showed how loose the Sperner's bound is, as well as a first pathway to improve upon it by considering the connections imposed by the homogeneous system of equations (9). The same example provides reasons to consider non-negative semiflows of minimal support. With that said, a comprehensive complexity analysis of the complete process of computing a generating set along with analyzing a Petri Net properties is required in order to quantitatively evaluate the trade-off between manipulating a generating set over $\mathbb{Q}^+$ or over $\mathbb{Q}$. Let us also point out that in the literature, there exist additional reasons to consider minimal supports described, for instance in [CTSH03].

We claimed that results in [CMPAW09] or in [CST03] p. 68 need to be rephrased in considering the formulation by the same authors in one of their previous publications [STC98].

We believe that these results may be enriched along two different alleys. From a mathematical point of view, the relation with integer linear programming or convex geometry has been investigated many times, particularly in [CS89], however, we believe it could be fruitful to look at the notion of toric varieties and saturated semigroups [Oda12]. From a Petri Net and, even more broadly, from a transition system theory point of view, applying these new results to a variety of models (for example, colored Petri Nets or any transition system that can be associated with a system of equations such as (9)) remains to be done.

Finally, we would like to stress the possibility to automate the proof scheme exhibited throughout the example of Section 7.3. Even though some proof steps can be shortened, we chose to develop our proofs in a systematic way to provide a clear road map on how an algorithm could proceed for solving such properties.

9. References

References

[AB86] N. Alon and K. A. Berman. Regular hypergraphs, gordon's lemma, steinitz' lemma and invariant theory. *J. of Combinatorial Theory, A*, 43:pp. 91–97, 1986.

[AM82] H. Alaiwan and G. Memmi. Algorithmes de recherche des solutions entières positives d'un système linéaire d'équations homogènes. *Revue Technique Thomson-CSF*, 14(1):pp. 125–135, 1982.

[Ash75] E.A. Ashcroft. Proving assertions about parallel programs. *Journal of Computer and System Sciences*, 10(1):pp. 110–135, 1975.

[BEI$^+$20] M. Bozga, J. Esparza, R. Iosif, J. Sifakis, and C. Welzel. Structural invariants for the verification of systems with parameterized architectures. In A. Biere and D. Parker, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 228–246, Cham, (2020). Springer Int. Pub.

[BFM18] M. Blondin, A. Finkel, and P. McKenzie. Handling infinitely branching well-structured transition systems. *Inf. Comput.*, 258:pp. 28–49, 2018.

[Bra82] G. W. Brams. *Réseaux de Petri: Théorie et Pratique.* Masson, Paris, France, 1982.

[CM88] K. Chandy and J. Misra. *Parallel Program Design, A Foundation.* Addison-Wesley, 1988.

[CMPAW09] G. Ciardo, G. Mecham, E. Paviot-Adet, and M. Wan. P-semiflow computation with decision diagrams. In Wolf K. (eds) Franceschinis G., editor, *ATPN, Petri Nets 2009*, LNCS 5606, pages 143–162. Springer, Berlin, Heidelberg, (2009).

[CS89] 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. In G. Rozenberg, editor, *Advances in Petri Nets 1990 [10th Int. Conf. on ATPN, Bonn, Germany, Proc.]*, LNCS 483, pages 79–112. Springer, 1989.

[CST03] J. M. Colom, M. Silva, and E. Teruel. Properties. In C. Girault and R. Valk, editors, *Petri Nets for Systems Engineering: A Guide* *to Modeling, Verification, and Applications*, pages 53–72. Springer Berlin Heidelberg, 2003.

[CTSH03] J. M. Colom, E. Teruel, M. Silva, and S. Haddad. Structural methods. In C. Girault and R. Valk, editors, *Petri Nets for Systems Engineering, A guide to Modeling, Verification, and Applications*, pages 277–316. Springer Berlin Heidelberg, 2003.

[Die17] R. Diestel. *Infinite Graphs*, pages 209–281. GTM 173. 5^{th} Edition, Springer, (2017).

[Dij71] E. W. Dijkstra. *Hierarchical Ordering of Sequential Processes. Acta Informatica*, 1:pp. 115–138, 1971.

[Dij75] E. W. Dijkstra. Guarded commands, non-determinacy, and formal derivation of programs. *CACM*, 18:pp. 453–457, 1975.

[DJ90] N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, pages 243–320. Elsevier and MIT Press, (1990).

[DL16] L. W. Dworzanski and I. A. Lomazova. Structural place invariants for analyzing the behavioral properties of nested petri nets. In F. Kordon and D. Moldt, editors, *ATPN and Concurrency*, pages 325–344, Cham, 2016. Springer Int. Pub.

[DRvB01] G. Delzanno, J. F. Raskin, and L. van Begin. Attacking symbolic state explosion. In *Proc. of the* 13^{th} *International Conference on Computer Aided Verification, CAV 2001*, LNCS 2102, pages 298–310. Springer, (2001).

[EH86] E. A. Emerson and J. Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *J. ACM*, 33(1):pp. 151–178, 1986.

[Eil74] S. Eilenberg. *Automata, Languages, and Machines*, volume A. Academic Press, 1974.

[Fin93] A. Finkel. The minimal coverability graph for petri nets. *in W. Reisig and G. Rozenberg (Eds) Advances in Petri Nets, LNCS 674*, pages pp. 210–243, 1993.

[Fin20] A. Finkel. From well structured transition systems to program verification. In L. Fribourg and M. Heizmann, editors, *Proc. 8th Int. Workshop on Verification and Program Transformation, VPT/HCVS@ETAPS 2020, Dublin, Ireland*, volume 320 of *EPTCS*, pages 44–49, 2020.

[FM83] A. Finkel and G. Memmi. Fifo nets: a new model of parallel computation. In A. B. Cremers and H.-P. Kriegel, editors, *TCS, 6th GI-Conference, Dortmund, Germany, January 5-7, 1983, Proc.*, volume 145 of *LNCS*, pages 111–121. Springer, 1983.

[FS01] A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1-2):pp. 63–92, 2001.

[GAH$^+$93] J. Gintell, J. Arnold, M. Houde, J. Kruszelnicki, R. McKenney, and G. Memmi. Scrutiny: A collaborative inspection and review system. In I. Sommerville and Manfred Paul, editors, *Software Engineering — ESEC '93*, pages 344–360, Berlin, Heidelberg, 1993. Springer.

[Gen87] H. J. Genrich. *Predicate/transition nets*, pages 207–247. Springer Berlin Heidelberg, 1987.

[GK18] R. Guerraoui and P. Kuznetsov. *Algorithms for Concurrent Systems.* EPFL Press, 2018.

[GV03] C. Girault and R. Valk. *Petri Nets for Systems Engineering, A guide to Modeling, Verification, and Applications.* Springer, 2003.

[Hoa15] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall Int., 2015.

[HT87] T. Hafer and W. Thomas. Computation tree logic ctl* and path quantifiers in the monadic theory of the binary tree. *Ottmann T. (eds), Automata, Languages and Programming. ICALP 1987. LNCS*, 267:pp. 269–279, 1987.

[JACB18] M. D. Johnston, D. F. Anderson, G. Craciun, and R. Brijder. Conditions for extinction events in chemical reaction networks with discrete state spaces. *J. Mathematical Biology*, 76,6:pp. 1535—-1558, 2018.

[JL22] P. Jančar and J. Leroux. Semilinear home-space is decidable for petri nets. *arXiv 2207.02697*, 2022.

[JR91] K. Jensen and G. Rozenberg. *High-level Petri Nets, Theory and Application.* Springer, 1991.

[JSS20] M. Jasper, M. Schlüter, and B. Steffen. Characteristic invariants in hennessy–milner logic. *Acta Informatica*, 57, 10 2020.

[Kel72] R. M. Keller. *Vector Replacement Systems: A formalism for Modeling Asynchronous Systems.* TR 117, Princeton U., Princeton, NJ, USA, December 1972.

[Kel76] R. M. Keller. Formal verification of parallel programs. *Commun. ACM*, 19(7):pp. 371–384, July 1976.

[KJ87] F. Krückeberg and M. Jaxy. Mathematical methods for calculating invariants in petri nets. In G. Rozenberg, editor, *Advances in Petri Nets 1987*, pages 104–131. Springer Berlin Heidelberg, 1987.

[KM69] R. Karp and R. Miller. Parallel program schemata. *J. of Computer and System Sciences*, vol. 3, issue 2:pp. 147–195, 1969.

[Kri63] S. A. Kripke. Semantical considerations on modal logic. *Acta Philosophica Fennica*, 16:pp. 83–94, 1963.

[Lan02] S. Lang. *Algebra,* 3^{rd} *revised Ed.* GTM, Springer, 2002.

[Ler09] J. Leroux. The general vector addition system reachability problem by presburger inductive invariants. In *Proc. of the* 24^{th} *Annual IEEE Symp. on Logic in Computer Science (LICS)*, pages 4–13. Also in LMCS Vol. 6 (3:22) 2010, pp. 1—-25. IEEE, (2009).

[Lie73] Y. E. Lien. Analysis of conservational transition systems. *ACM '73: Proc. of the ACM annual conference*, 5(2):pp 440.4–441, August 1973.

[Lie76a] Y. E. Lien. A note on transition systems. *Information Sciences*, 19:pp. 347–362, 1976.

[Lie76b] Y. E. Lien. Termination properties of generalized petri nets. *SIAM J. on Computing*, 5(2):pp. 251–265, june 1976.

[LM89] J. B. Lasserre and P. Mahey. Using linear programming in petri net analysis. *RAIRO, RO*, 23(1):pp 43–50, 1989.

[LS74] K. Lautenbach and H. A. Schmid. Use of petri nets for proving correctness of concurrent process systems. In *Information Processing'74, Proc. I.F.I.P. Congress*, pages 187–191. North Holland Pub., (1974).

[Mem77] G. Memmi. Semiflows and invariants. application in petri nets theory. *in Journées d'Etudes sur les Réseaux de Petri AFCET-Institut de Programmation)*, pages pp. 145–150, 1977.

[Mem78] G. Memmi. *Fuites et Semi-flots dans les Réseaux de Petri.* Thèse de Docteur-Ingénieur, U. P. et M. Curie, Paris, France, 1978.

[Mem83] G. Memmi. *Methodes d'analyse de Réseaux de Petri, Réseaux a Files, Applications au temps reel.* Thèse d'Etat, U. P. et M. Curie, Paris, France, 1983.

[Mem19] G. Memmi. *How Carl Adam Petri Deeply Influenced My Understanding of Invariance and Parallelism*, pages 133–139. in W. Reisig and G. Rozenberg (Eds), Carl Adam Petri: Ideas, Personality, Impact, Springer Int. Pub., Cham, Switzerland, (2019).

[Mem23] Gerard Memmi. Minimal generating sets for semiflows. In Marieke Huisman and António Ravara, editors, *Formal Techniques for Distributed Objects, Components, and Systems*, pages 189–205, Cham, 2023. Springer Nature Switzerland.

[MM81] R. Martin and G. Memmi. Specification and validation of sequential processes communicating by fifo channels. In *In: 4^th Int. Conf. on Software Engineering for Telecommunication Switching Systems, Warwick U. Conventry, U.K*, pages 54–57. SIEE, 1981.

[MOSS99] M. Müller-Olm, D. A. Schmidt, and B. Steffen. Model-checking: A tutorial introduction. *In Static Analysis 6^{th} Int Symposium SAS 99*, LNCS 1694:pp. 330–354, 1999.

[MS82] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalised petri net. In *ATPN*, pages 301–310. Springer, 1982.

[MVD03] R. Mackenthun, M. Voorhoeve, and A. Diagne. Case studies. In C. Girault and R. Valk, editors, *Petri Nets for Systems Engineering, A guide to Modeling, Verification, and Applications*, pages 159–178. Springer Berlin Heidelberg, 2003.

[Oda12] T. Oda. *Convex Bodies and Algebraic Geometry (An Introduction to the Theory of Toric Varieties).* Springer, 2012.

[Pet96] Carl A. Petri. Nets, time, and space. *TCS*, 153(1):pp 3–48, 1996.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. of the* 18th *Annual Symposium on Foundations of Computer Science*, SFCS '77, pages 46–57. IEEE, (1977).

[Ram73] C. Ramchandani. *Analysis of asynchronous concurrent systems by timed Petri nets.* Dpt of Electrical Engineering, MIT, Cambridge, MA, USA, 1973.

[Ray86] M. Raynal. *Algorithms for mutual exclusion.* The MIT Press, 1986.

[RT19] M. Raynal and G. Taubenfeld. Fully anonymous shared memory algorithms. *arXiv 1909.05576*, 2019.

[Sak09] J. Sakarovitch. *Elements of Automata Theory.* Cambridge University Press, 2009.

[Sch87] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley, 1987.

[Sen10] L. A. Seneca. *Natural Questions.* translated by H. M. Hine, University of Chicago Press, Chicago, USA, 2010.

[Sif78] J. Sifakis. Structural properties of petri nets. In J. Winkowski, editor, *MFCS 1978, Proc.* 7th *Symp., Zakopane, Poland*, volume 64 of *LNCS*, pages 474–483. Springer, (1978).

[Spe28] E. Sperner. Ein satz *über* untermengen einer endlichen menge. *Mathematische Zietschrift*, 27:pp. 544–548, 1928.

[SSM03] S. Sankaranarayanan, H. Sipma, and Z. Manna. *Petri Net Analysis Using Invariant Generation*, pages 682–701. N. Dershowitz, *Verification: Theory and Practice: Essays Dedicated to Zohar Manna on the Occasion of His* 64th *Birthday*, Springer, Berlin, Heidelberg, 2003.

[STC96] M. Sylva, E Teruel, and J. M. Colom. Linear algebraic and linear programming techniques for the analysis of places/transitions net systems. *in W. Reisig and G. Rozenberg (Eds) Lectures on Petri Nets I: Basic Models. ACPN 1996, LNCS 1491*, pages pp. 309–373, 1996.

[STC98] M. Silva, E. Teruel, and J. M. Colom. *Linear algebraic and linear programming techniques for the analysis of place/transition net systems*, pages 309–373. Springer Berlin Heidelberg, 1998.

[Tou81] J.M. Toudic. *Algorithmes d'Analyse structurelle des Réseaux de Petri.* Thèse de 3^{eme} cycle, U. P. et M. Curie, Paris, France, 1981.

[TRSS01] A. Tiwari, H. Rueß, H. Saïdi, and N. Shankar. A technique for invariant generation. In T. Margaria and W. Yi, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 113–127, Berlin, Heidelberg, (2001). Springer.

[Val78] R. Valk. Self-modifying nets, a natural extension of petri nets. In G. Ausiello and C. Böhm, editors, *Automata, Languages and Programming, 5^{th} Colloquium, Udine, Italy, Proc.*, volume 62 of *LNCS*, pages 464–476. Springer, 1978.

[Val81] R. Valk. Generalizations of petri nets. In *MFCS'81, LNCS 118*, pages 140–155. Springer Verlag, (1981).

[VM84] J. Vautherin and G. Memmi. Computation of flows for unary-predicates/transition-nets. In G. Rozenberg, H. Genrich, and G. Roucairol, editors, *Advances in Petri Nets 1984, European Workshop on Applications and Theory in Petri Nets, selected papers*, volume 188 of *LNCS*, pages 455–467. Springer, 1984.

[Wol19a] K. Wolf. *How Petri Net Theory Serves Petri Net Model Checking: A Survey*, pages 36–63. Springer Berlin Heidelberg, 2019.

[Wol19b] K. Wolf. *How Petri Net Theory Serves Petri Net Model Checking: A Survey.* In M. Koutny, L. Pomello, and L. M. Kristensen, editors, *Transactions on Petri Nets and Other Models of Concurrency XIV*, pages pp. 36–63, Berlin, Heidelberg, 2019. Springer Berlin Heidelberg.