CHAPTER 1

PETRI NETS

1 An introduction

An alternative to automata for untimed models of DES is provided by Petri nets (PNs). These models were first developed by C. A. Petri in the early 1960s. As we will see, Petri nets are related to automata in the sense that they also explicitly represent the transition function of DES. Like an automaton, a Petri net is a device that manipulates events according to certain rules. One of its features is that it includes explicit conditions under which an event can be enabled; this allows the representation of very general DES whose operation depends on potentially complex control schemes. This representation is conveniently described graphically, at least for small systems, resulting in Petri net graphs; Petri net graphs are intuitive and capture a lot of structural information about the system. We will see that an automaton can always be represented as a Petri net; on the other hand, not all Petri nets can be represented as finite-state automata. Consequently, Petri nets can represent a larger class of languages than the class of regular languages, R. Another motivation for considering Petri net models of DES is the body of analysis techniques that have been developed for studying them. These techniques include reachability analysis, similarly to the case of automata, as well as linear-algebraic techniques. Such techniques cover not only untimed Petri net models but timed Petri net models as well; in this regard, we will see in the next chapter that there is a well-developed theory, called the "max-plus algebra," for a certain class of timed Petri nets (cf. Sect. 5.4). Finally, we mention that control of Petri nets is an active research area and there are controller synthesis techniques that exploit the structural properties of Petri nets.

2 A *static* view of a PN

The process of defining a Petri net involves two steps. First, the Petri net graph, also called Petri net structure, which is analogous to the state transition diagram of an automaton. Then we will define the *dynamic* behind this graph by characterizing the concept of state, related to **marking** (non to be confused with marked states when referring to automata), and a transition labeling function, resulting in the complete Petri net model.

2.1 PN characterization

In Petri nets, events are associated with transitions. In order for a transition to occur, several conditions may have to be satisfied. Information related to these conditions is contained in places. Some such places are viewed as the "input" to a transition; they are associated with the conditions required for this transition to occur. Other places are viewed as the output of a transition; they are associated with conditions that are affected by the occurrence of this transition. Transitions, places, and certain relationships between them define the basic components of a Petri net graph. A Petri net graph has two types of nodes, places and transitions, and arcs connecting these. It is a bipartite graph in the sense that arcs cannot directly connect nodes of the same type; rather, arcs connect place nodes to transition nodes and transition nodes to place nodes. The precise definition of Petri net graph is as follows.

Definition 2.1. An **unmarked** Petri Net is a triplet

$$(P, T; A) \tag{1.1}$$

where P is the *place set*, T is the *transitions set* and A the *arc set* characterizing a *flow relationship* between places and transitions as follows:

- $P \cap T = \varnothing$ (places and transitions cannot be *mingled*);
- $P \cup T \neq \varnothing$ (a PN must have at least a place or a transition);
- $A \subseteq (P \times T) \cup (T \times P)$ (arcs could connect places to transitions or transitions to places, no places vs places or transitions vs transitions arcs are admissible in a PN).

We will normally represent the set of places by $P = \{p_1, p_2, \ldots, p_n\}$, and the set of transitions by $T = \{t_1, t_2, \ldots, t_m\}$; thus, in the remainder of this chapter, $|P| = n$ and $|T| = m$. A typical arc is of the form (p_i, t_j) or (t_j, p_i). Note that we could allow P and T to be countable, rather than finite sets, as is the case for the state set in automata. It turns out, however, that a finite number of transitions and places is almost always perfectly adequate in modeling DES of interest. In addition a PN, where the graph nodes (places and transitions) are *distinct* and the arcs are allowed only to connect places and transitions or transitions and places is a particular form of *bipartite graph*. From a graphical point of view places are denoted by rounded circles and transition rectangles. The following example shows a PN with three places and a single transition

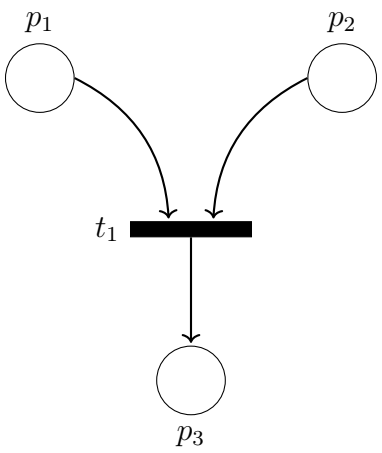

Figure 1.1: Petri Net first example

In order to describe the flow relationship between places and transitions and to understand how the network changes its state behavior the notion of *Pre-* and *Post-* is crucial.

Definition 2.2. Let $X = P \cup T$, then the *preset* of a place (transition) is the following set of transitions (places)

$$Pre(\alpha) = {}^\bullet\alpha = \{\delta \in X \mid (\delta,\ \alpha) \in A\} \tag{1.2}$$

Definition 2.3. Let $X = P \cup T$, then the *postset* of a place (transition) is the following set of transitions (places)

$$Post(\alpha) = \alpha^\bullet = \{\delta \in X \mid (\alpha,\ \delta) \in A\} \tag{1.3}$$

As an example when considering the PN in Figure 1.1 we have

$${}^\bullet t_1 = \{p_1,\ p_2\},\quad t_1{}^\bullet = \{p_3\},\quad {}^\bullet p_1 = \varnothing,\quad p_1{}^\bullet = \{t_1\}$$

$${}^\bullet p_2 = \varnothing,\quad p_2{}^\bullet = \{t_1\},\quad {}^\bullet p_3 = \{t_1\},\quad p_3{}^\bullet = \varnothing$$

Until now we have focused only on the *topological* aspect of the network and additional ingredients are necessary to understand how a PN can describe the evolution of a DES (dynamic perspective).

3 Places-Transition Petri Network

In what follows, we will introduce token/marking concept for a given PN

Definition 3.1. A Places-Transition Petri Network is a quintuple

$$\mathcal{N} = (P,\, T;\, A,\, W,\, M_0) \tag{1.4}$$

with places, transitions and arc sets compliant with Definition 2.1 and

1. $W : A \to \mathbb{N}\backslash\{0\}$, a function assigning a given strictly positive *weight* to each arc of the PN;

2. $M_0 : P \to \mathbb{N}$, an initial *marking* function assigning an integer called *number of tokens*, to each place of the PN.

The weight is represented on each arc with an integer number and, in case of unitary weight the notation is omitted. A Petri net wher all arc weights is unitary is named *ordinary*. The attribute *initial* in the marking suggests that the network is going to exhibit an evolution in terms of tokens appearing, disappearing or increment at each place of the network: the behavior is governed by rules essentially related to the transitions status. Thus, the initial marking function defines a row vector $x_0 = [M_0(p_1),\, M_0(p_2), \ldots, M_0(p_n)]$, where n is the number of places in the Petri net. The i-th entry of this vector indicates the (non-negative integer) initial number of tokens in place p_i. In Petri net graphs, a token is indicated by a dark dot positioned in the appropriate place. Now, since our system modeling efforts have always relied on the concept of state, in the case of a Petri net we identify the marking with the state of the Petri net. That is, we define the state of a Petri net to be its marking row vector $x = [M(p_1),\, M(p_2), \ldots, M(p_n)]$ by indicating with

$$M : P \to \mathbb{N}$$

the (not necessarily initial) *marking* function assigning a given number of tokens to each place at the generic event. Note that the number of tokens assigned to a place is not necessarily bounded. It follows that the number of states we can have is, in general, infinite. Thus, the state space of a Petri net with n places is defined by all n-dimensional vectors whose entries are non-negative integers, that is, $\mathbb{N}^n$. While the term *marking* is more common than *state* in the Petri net literature, the term state is consistent with the role of state in system dynamics, as we shall now see; moreover, the term state avoids the potential confusion between marking in Petri net graphs and marking in the sense of marked states in automata.

The above definitions do not explicitly describe the state transition mechanism of Petri nets. This is clearly a crucial point since we want to use Petri nets to model dynamic DES. It turns out that the state transition mechanism is captured by the structure of the Petri net graph. In order to define the state transition mechanism, we first need to introduce the notion of enabled transition. Basically, for a transition $t \in T$ to "happen" or to "be enabled," we require a token to be present in each place (i.e., condition) which is input to the transition. Since, however, we allow weighted arcs from places to transitions, we use a slightly more general definition.

Definition 3.2. Given a Place-Transition Petri Network $\mathcal{N}$ with an arbitrary state

$$x = [M(p_1),\, M(p_2), \ldots, M(p_n)]$$

The transition $t \in T$ is *enabled to fire* (*enabled* for shorten) when

$$M(p) \geq W((p,\, t)),\ \forall p \in \bullet t \tag{1.5}$$

A compact writing to state that the transition t is enabled is

$$M\,[t>$$

In words, the transition t in the Petri net is enabled when the number of tokens in p is at least as large as the weight of the arc connecting p to t , for all places p that are in the Preset of t.

In the following example we can obserbe a Petri Net whose arc weights are as follows:

$$W((p_1,\ t_1)) = 2,\ W((p_2,\ t_1)) = 1,\ W((p_1,\ t_1)) = 3$$

and the inital marking is

$$x = [2,\ 3,\ 0]$$

transition t_1 is enabled according to (1.5).

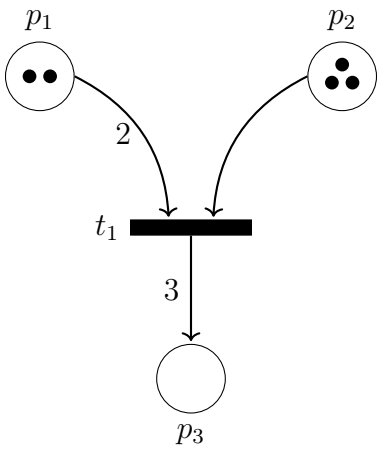

Figure 1.2: Transition t_1 enabled

As was mentioned above, since places are associated with conditions regarding the occurrence of a transition, then a transition is enabled when all the conditions required for its occurrence are satisfied; tokens are the mechanism used to determine satisfaction of conditions. The set of enabled transitions in a given state of the Petri net is equivalent to the active event set in a given state of an automaton. We are now ready to describe the dynamical evolution of Petri nets.

Definition 3.3. Given the state of a Place-Transition Petri Network $\mathcal{N}$

$$x = [M(p_1),\ M(p_2), \ldots, M(p_n)]$$

The *firing* of an enabled transition $t \in T$ generates a new state of the Network

$$x' = [M'(p_1),\ M'(p_2), \ldots, M'(p_n)]$$

under the following rules

1. $\forall p \in {\bullet t} \backslash t{\bullet}$ (all places in the Preset but not in the Postset of t)

$$M'(p) = M(p) - W((p,\ t)) \tag{1.6}$$

2. $\forall p \in t\bullet \setminus \bullet t$ (all places in the Postset but not in the Preset of t)

$$M'(p) = M(p) + W((t, p)) \tag{1.7}$$

3. $\forall p \in \bullet t \cap t\bullet$ (places belonging both to the Postset and the Preset of t)

$$M'(p) = M(p) - W((p, t)) + W((t, p)) \tag{1.8}$$

4. otherwise, $M'(p) = M(p)$.

The definition meaning is easy, in the first case since the node p belong to the Preset he will *lose* a number of tokens equal to the arc weight connecting the node to the firing transition. In the second case since the node belong to the Postset he will *gain* a number of tokens equal to the arc weight connecting the node to the firing transition. In the third case there will a tokens *balance* between arc weights entering the transition (losing) and the arc weights leaving the transition (gain). In the last case the marking is left unchanged.

Notice that the transition firing is a *local* event related to the Network configuration. It can be in fact observed that all the places where the marking changes are only those directly tied to the transition in terms of Pre- and Post- sets. All the other places are left unchanged in term of marking.

In the following example we will analyze the firing of transition t_1

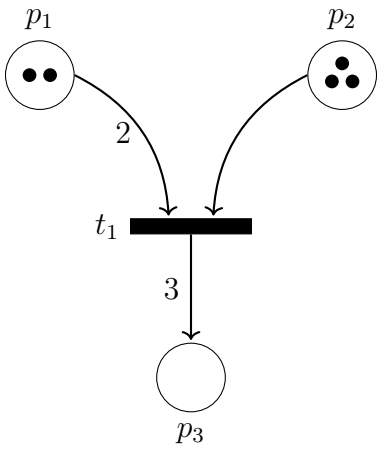

Figure 1.3: Before the firing of t_1

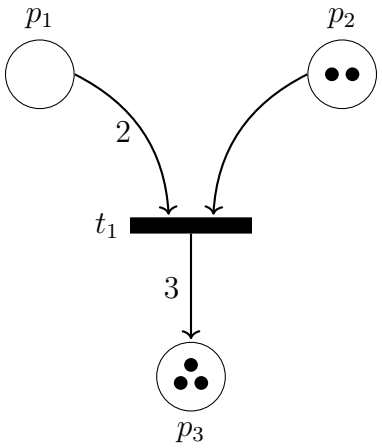

Figure 1.4: After the firing of t_1

It is easy to observe that p_1 and p_2 are losing a number of tokens equal to the weight of the arcs connecting the places to t_1 and p_3 is gaining a number of tokens equal to the weight of the arc connecting t_1 to p_3. As a consequence the initial state is

$$x = [2,\ 3,\ 0]$$

and the state after the firing is

$$x' = [0,\ 2,\ 3]$$

t_1 is no longer enabled after the firing.

3.1 Non Deterministic PN feature

It is worth noticing that in a Petri Net it should be possibile that more than a transition could be enabled, only one transition at a time fires, and the following question arises: which, among all the enabled transitions, is allowed to fire? The answer is that the transition allowed to fire is only one picked via a random choice (*rolling dice* fashion). This strategy allows to preserve the *local* features of a Petri Net (only the places belong to the Pre- a Post-sets change their marking). In addition, after the *randomly picked* transition has fired it should be possible that some transitions should be disabled and new transition enabled, as a consequence at the next step a new *random* evaluation of the newly enabled transitions scenario is mandatory.

3.2 Standard PN scenarios

Transitions in sequence

Two transitions t_1 and t_2 are in sequence t_1 i.e. precedes t_2 in a given marking scenario if t_1 is enabled and the firing of t_1 enables t_2.

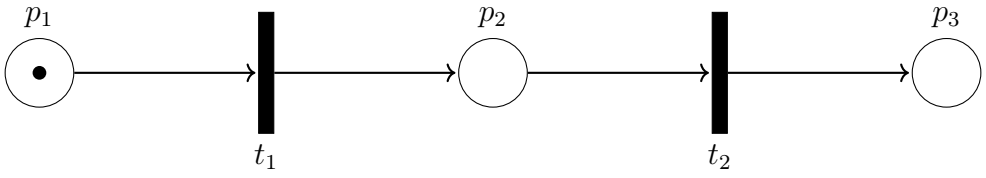

Figure 1.5: t_1 and t_2 in sequence (t_1 enabled in the current marking)

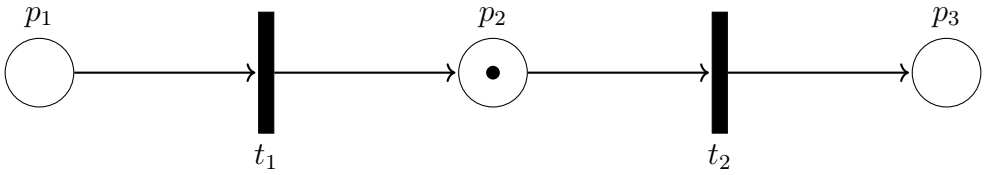

Figure 1.6: t_1 and t_2 in sequence (t_2 enabled after the firing of t_1)

Conflicting transitions

Two transitions t_1 and t_2 are in *structural conflict* only if they share a unique place. The conflict becomes *effective* when both transitions are enabled in a given marking scenario but the number of tokens is not sufficient to leave one of the two transitions enabled after the firing of the other (remember that the transition to be fired is randomly chosen at each step). It is obvious to notice that *effective conflict* is implied by *structural conflict*.

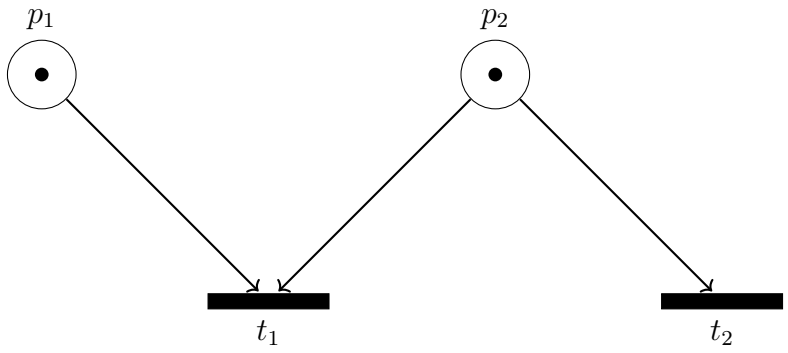

Figure 1.7: t_1 and t_2 in conflict (p_1 is shared between the two transitions)

Concurrency between transitions

Two transitions t_1 and t_2 are *structurally concurrent* if they do not share places. The concurrency becomes *effective* when both transitions are enabled in a given marking scenario and the firing of a transition (remember that the transition to be fired is randomly chosen at each step) leaves unchanged the enabling status of the other. It is obvious to notice that *effective concurrency* is implied by *structural concurrency.*

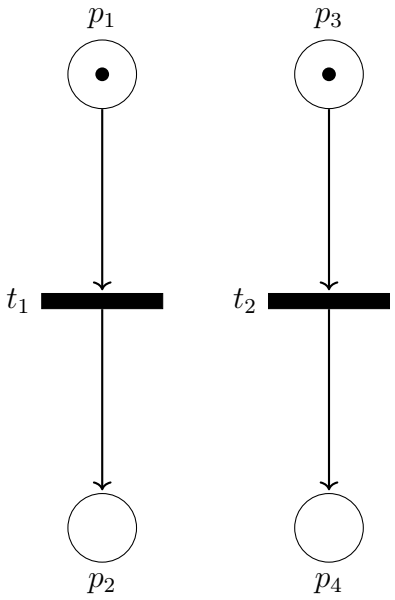

Figure 1.8: Concurrent transitions t_1, t_2

If a transition shares two places we are facing a *syncronization* scenario since the enabling requires sufficent tokens in both places.

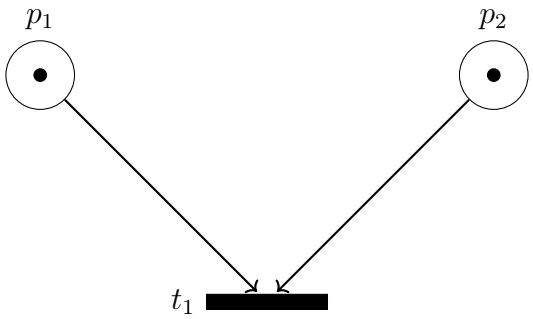

Figure 1.9: t_1 synchronizes p_1 and p_2

An enabled transition is in a *concurrency start* scenario whenever two (or more) concurrent sequences are activated by the transition firing

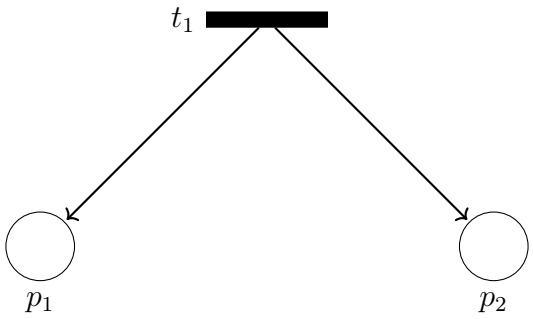

Figure 1.10: t_1 start a concurrent phase on p_1 and p_2

3.3 Petri Net main properties

In what follows we will state some properties related to Petri Nets that are of importance when characterizing the behavior of industrial/manufacturing processes described by this framework. It is of paramount relevance to notice that formal results regarding the check of these properties are lacking. Nonetheless for particular PN classes some (even partial) results exist and must be stated and analyzed.

Reachability

We will start with the observation about the dynamic behavior of Petri Nets that not all states in $\mathbb{N}^n$ can necessarily be reached from a Petri net graph with a given initial state characterized by the marking M_0.

Definition 3.4. A marking M_1 in a Petri Net is said to be *reachable* from a given marking M if there exists a sequence of transitions such that, starting from M_1 and if fired one after another, are capable to achieve the marking M_1 in the net at the end of the sequence.

The reachability problem is of paramount relevance to determine the behavior of the PN. It is unfortunately not *easy* to be solved by stating simple and general results. In what follows the set of reachable markings from a given initial marking M will be denoted as $[M >$.

Reversibility and Home State

A lot of manufacturing/industrial process require the possibility to recover the normal initial working operation after malfunction and or fault occurrences. Also, it is requested to revert to the *initial state* after a sequence of operations have been accomplished.

Definition 3.5. A Petri Net is *reversible* if, for each $M \in [M_0 >$, M_0 is reachable from M.

Definition 3.6. A marking M_i of a Petri Net is *home state* if, for each $M \in [M_0 >$, M_i is reachable from M.

Boundedness

Petri Nets may describe manufacturing systems where consumers, producers and goods play a significant role in terms of tokens distributed along certain places. It is important to check if all the entities involved in these PN have limited values along arbitrary firing sequences of enabled transitions. Also, control strategies design for these DES require to monitor goods/materials stockpile in storage location to avoid congestion.

Definition 3.7. A place in a Petri Net is called *k-limited* if, for all reachable markings in the net, the number of tokens in the place does not exceed a given integer k.

Definition 3.8. A Petri Net is *k-limited* if all places are k-limited. The net is *limited* if is *k-limited* for a certain (not a-priori fixed) k.

If a Petri Net is limited all the possible markings (states) are limited and is equivalent to a Finite State Automaton. In figure 1.11 an example of a non-limited Petri Net is provided,

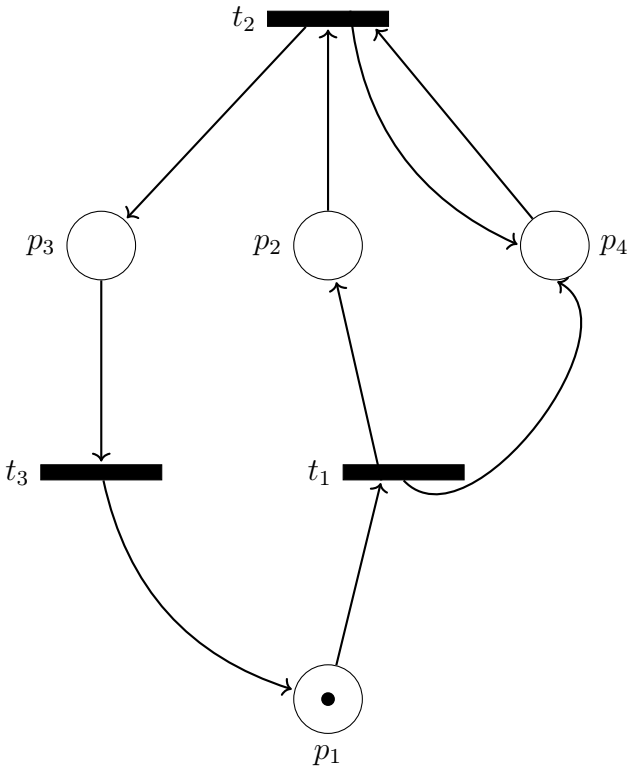

Figure 1.11: PN not limited

notice that the transitions sequence t_1, t_2, t_3, when repeated, is going to generate an endless increasing number of tokens on the place p_4.

A particular limited PN category is when $k = 1$, in this case we obtain a *safe* network. Safe networks are characterized by an initial marking M_0 characterized by a binary vector and all the reachable markings are binary vectors. With a safe net we describe simple DES, as an example on-off controllers, systems describing free/busy situations etc.

Liveness

Liveness in a Petri Net implies the possibility that an arbitrary transition it can always eventually fire, regardless of the current state of the net.

Definition 3.9. A transition $t \in T$ in a Petri Net is *alive* if and only if $\forall M,\ M \in [M_0 >$, $\exists M^* \in [M >$ such that t is enabled under M^*.

Definition 3.10. A Petri Net is *alive* if and only if their transitions are alive.

In words a transition t is alive if, starting from an initial marking M_0, all the possible reachable markings make it possible to drive the network to a marking M^* where the transition is enabled. This implies that M^* belongs to the reachable markings from M_0 and, according to the definition we will be able to drive the network to a new marking, named M^{**} where t is enabled for the second time. It easy to understand that an alive transition can fire an infinite number of times.....

Clearly, liveness is a very stringent condition on the behavior of the system. Moreover, checking for liveness as defined above is an extremely tedious process, often practically infeasible for many systems of interest.

Definition 3.11. A marking M in a Petri Net is *alive* if and only if $\forall t \in T$, $\exists M^* \in [M >$, where t is enabled.

On the basis of the previous definition from an alive marking is possible to fire each transition of the network. Nonetheless not all markings in a PN enjoy this feature.

Proposition 3.1. A Petri Net is alive if and only if all markings reachable from the initial marking are alive.

Definition 3.12. A marking in a Petri Net is *dead* if and only if no transition is enabled under M.

Reversibility, Boundedness and Liveness are independent each other and as a consequence 8 types of Petri Nets are possible by combining the three properties (and their negations).

3.4 State Equations

Let us reconsider the situation describing how the state value of an individual place changes when a transition fires. It is not difficult to see how we can generate a vector equation from the rules described in Definition 3.2, in order to specify the next Petri net state

$$x' = [M'(p_1),\ M'(p_2), \ldots, M'(p_n)]$$

given the current marking described by the state

$$x = [M(p_1),\ M(p_2), \ldots, M(p_n)]$$

giveng the fact that a particular enabled transition, t_j , has fired. To do so, let us first define the firing vector u, an m-dimensional row vector of the form

$$u = [0, 0, \ldots, 0, 1, 0, \ldots, 0]$$

where the 1 is located in the j-th position. By computing the *incidence matrix* of the Petri Net $\mathbf{A} \in \mathbb{Z}^{m \times n}$ where each element is

$$a_{ji} = W((t_j, p_i)) - W((p_i, t_j))$$

we finally obtain the state transition equation of the Petri Net

$$x' = x + u\,\mathbf{A} \tag{1.9}$$

which describes the state transition process as a result of an "input" u, that is, a particular transition firing. The state equation provides a convenient algebraic tool and an alternative to purely graphical means for describing the process of firing transitions and changing the state of a Petri net. The incidence matrix can be obtained by combining two rectangular matrices having the same dimension as the incidence matrix: the first one characterizes all the arcs *going out* from a given transition (Output Matrix, $\mathbf{O}$)

$$\mathbf{O}_{ji} = W((t_j, p_i))$$

and the second one characterizes all the arcs *going inside* a given transition (Input Matrix, $\mathbf{I}$)

$$\mathbf{I}_{ji} = W((p_i, t_j))$$

Then we have

$$\mathbf{A} = \mathbf{O} - \mathbf{I}$$

In particular, the *input matrix* could be used to verify if a given transition is enabled to fire under an arbitrary marking M. Remember that the marking characterizes the state of the network

$$x = [M(p_1),\ M(p_2), \ldots, M(p_n)]$$

and, by arranging the input matrix in a row fashion structure

$$\mathbf{I} = \begin{bmatrix} \mathbf{I}_1 \\ \mathbf{I}_2 \\ \vdots \\ \mathbf{I}_m \end{bmatrix}$$

transition t_j, $j = 1, \ldots, m$ is enabled if, in a componentwise fashion

$$x \geq \mathbf{I}_j$$

which implies that, for each place belonging to the Preset of t_j the number of tokens exceeds the weight of the arc connecting the place to the transition.

3.5 Firing Sequences

Let us suppose that a Petri Net is under the marking configuration M_1 i.e. the state is

$$x_1 = [M_1(p_1),\ M_1(p_2), \ldots, M_1(p_n)]$$

if a generic transition t_1 is enabled and fires the net is going to exhibit a new marking M_2 and the corresponding state

$$x_2 = [M_2(p_1),\ M_2(p_2), \ldots, M_2(p_n)] = x_1 + u_1\,\mathbf{A}$$

will arise. It is natural that under the new marking some transitions previously enabled could be disabled and viceversa. If a new enabled transition, say t_2 is going to fire the Network will show a new marking/state M_3 and so on. To compactly describe the state transition on an ordered firing succession the notion of *firing sequence* is introduced.

Definition 3.13. A *firing sequence* in M_0, $s = t_0, t_1, \ldots, t_l$, $l > 0$, is an ordered sequence of transitions where t_0 is enabled in M_0 and the firing of t_i, $0 \leq 1 \leq l-1$ enables t_{i+1}.

The following notation can be used to describe the previous definition when M_0 is the initial marking and M_l the marking after the ordered firings of $s = t_0, t_1, \ldots, t_l$

$$M_0\,[s > M_l$$

the final state

$$x_l = [M_l(p_1),\ M_l(p_2), \ldots, M_l(p_n)]$$

can then be obtained by iteratively applying the state equation

$$x + u\,\mathbf{A}$$

under the hypothesis that the vector u changes its *shape* on the basis of the current enabled transition. It is worth noticing that when a transition fires the following quantity

$$u\,\mathbf{A} = x' - x$$

difference between the new and the old state of the net does not depends on the *network state history*.

4 Petri Net Analysis

4.1 Reachability Tree/Graph

In order to have a comprehension of the *mechanism* which characterizes a Petri Net all the possible markings reachable from an initial state/marking M_0 need to be obtained. A *simple* solution can be implemented: consider, starting from the initial marking, the enabled transitions and determine the next marking/markings by randomly firing each other and so on. The collection of all these markings is the *PN Set of Reachable states*.

Definition 4.1. Given a Petri Net $\mathcal{N}$ with initial marking M_0 the *Set of Reachable States*, $R(\mathcal{N},\ M_0)$ is the smallest marking set such that

1. $M_0 \in R(\mathcal{N},\ M_0)$
2. $M^* \in R(\mathcal{N},\ M_0)$ and $\exists t \in T$ such that $M^*[t > M^{**} \Rightarrow M^{**} \in R(\mathcal{N},\ M_0)$

The definition is easy to be exaplained: the first condition states that the initial marking belong the Reachability Set and the second states that all the markings that are reachable from a given marking belonging to the Reachability Set belong to that set too. To better understand the *dynamics* of this set the *reachability tree* and/or *reachability graph* can be used for this purpose. The **Reachability Tree** (also known as a **Coverability Tree**) is a directed tree structure that represents all possible markings (or states) that the Petri Net can reach from a given initial marking. Each node in the tree corresponds to a reachable marking, and each edge corresponds to the firing of a transition that leads from one marking to another.

Key Features of the Reachability Tree

- **Initial Marking**: The root of the tree represents the initial marking of the Petri Net.
- **Transitions**: Each edge in the tree is labeled with a transition, representing the firing of that transition from one marking to reach a new marking.
- **Markings**: Each node represents a marking in which the number of tokens in each place is indicated.
- **Boundedness Check**: If a place's tokens become unbounded during exploration, this is represented with the symbol ω, indicating infinite token count (*piggybank place*).

The Reachability Tree provides a way to analyze various properties of the Petri Net, including:

- **Reachability**: Checking if a particular marking can be reached from the initial marking.
- **Boundedness**: Determining if there are places in the net where tokens can grow indefinitely.
- **Liveness**: Assessing if transitions can potentially become disabled (dead transitions) due to token limitations.

In the context of a Place-Transition (PT) Petri Net, a **Reachability Graph** is a directed graph that represents all possible markings (states) the Petri Net can reach from a given initial marking. Each node in the graph corresponds to a unique reachable marking, and each directed edge represents the firing of a transition that leads from one marking to another.

Key Features of the Reachability Graph

- **Initial Marking**: The starting node of the graph represents the initial marking of the Petri Net.
- **Transitions**: Each edge is labeled with the transition that fires to move from one marking to the next.
- **Finite Representation**: Unlike the Reachability Tree, the Reachability Graph contains only unique markings, meaning it has a finite number of nodes if the Petri Net is bounded.

- **Cycles and Loops**: The Reachability Graph may contain cycles, where a marking can return to a previous marking through a sequence of transitions.

Similarly as the Reachability Tree, the Reachability Graph is used to analyze key properties of the Petri Net, such as:

- **Reachability**: Determining if a specific marking can be reached from the initial marking.
- **Boundedness**: Checking if the Petri Net has a finite number of reachable markings (i.e., it is bounded).
- **Liveness**: Assessing if all transitions can eventually fire from some reachable marking, ensuring the system does not reach a deadlock state.
- **Deadlock Detection**: Identifying any markings from which no transitions are enabled, indicating a deadlock.

The Reachability Graph provides a compact representation of all reachable markings and transitions from the initial marking, allowing for comprehensive analysis of the dynamic properties of the Place-Transition Petri Net.

Let us start with a simple example The Initial Marking is as follows with the related

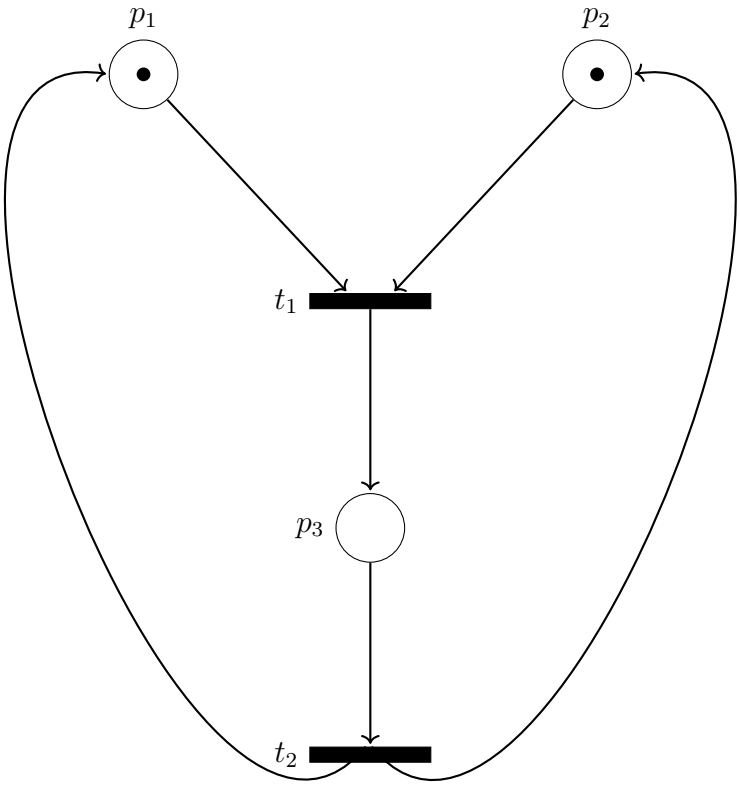

Figure 1.12: Simple PN

enabled transition

$$M_0 = [1,\ 1,\ 0]\,,\quad t_1 \text{ Enabled}$$

the t_1 firing generates the following PN configuration

$$M_0\,[t_1 > = M_1 = [0,\ 0,\ 1]\,,\quad t_2 \text{ Enabled}$$

the t_2 firing generates the following PN configuration

$$M_1\,[t_2 > = M_2 = M_0 = [1,\ 1,\ 0]\,,\quad t_1 \text{ Enabled}$$

Notice that the last marking is *identical* to the initial Marking and is named as *duplicate*. We have then exploited all the possible *reachability paths* originating from M_0 and obtained the Reachability Set that can be described by a Tree structure or Graph structure as follows.

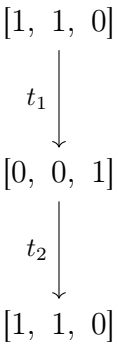

Figure 1.13: Reachability Tree

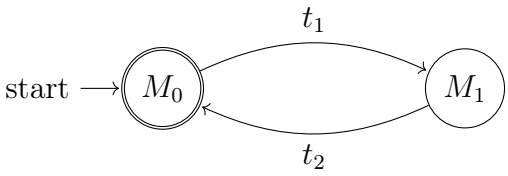

Figure 1.14: Reachability Graph

It must be noted that the PN is bounded because all nodes in reachability tree/graphs are denoted by *finite marking* vectors.

The *scenario* becomes tricky when generic (not necessarily bounded) Petri Nets are taken into consideration.

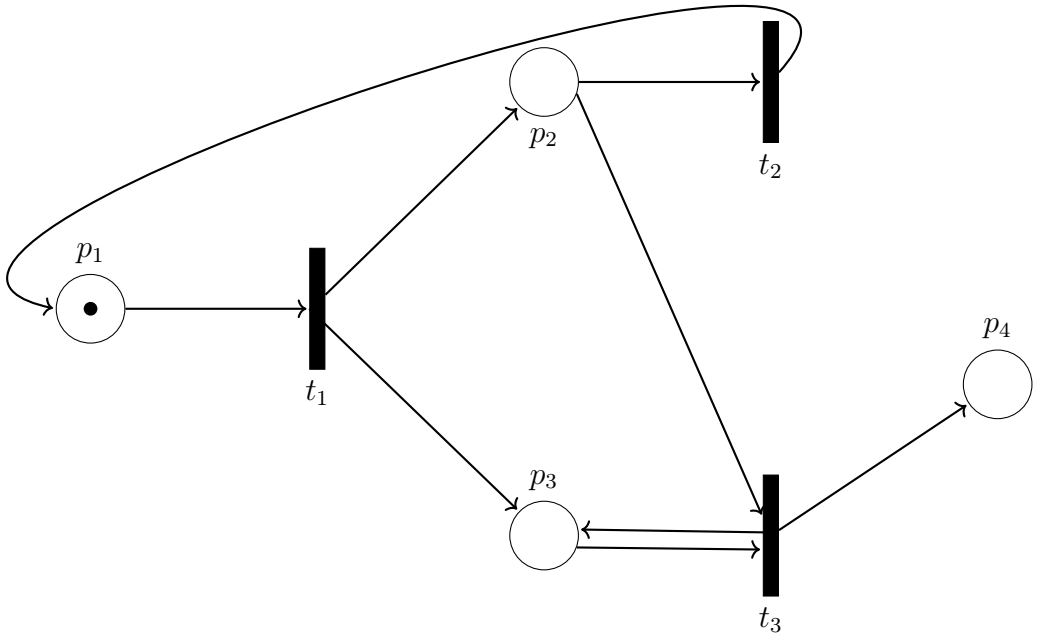

Figure 1.15: Unbounded PN

The initial marking and the enabled transition(s) are

$$M_0 = [1,\ 0,\ 0,\ 0],\ t_1 \ \text{Enabled}$$

the t_1 firing generates the following non-deterministic scenario on the enabled transitions

$$M_0\,[t_1 > = M_1 = [0,\ 1,\ 1,\ 0]\,,\quad t_2 \text{ Enabled and } t_3 \text{ Enabled}$$

the scenario on the Reachability tree is simple, if the markings obtained by the separate firings of the transitions are generating *new* markings, two branches will be generated from the leaf M_1, the first related to the firing of t_2 and the second to the firing of t_3 (the same holds true for the graph). Let us consider now the t_3 firing which generates a *simple* marking

$$M_1\,[t_3 > = M_2 = [0,\ 0,\ 1,\ 1]\,,\quad \text{no transitions Enabled}$$

the M_2 marking is a typical characterization of a dead marking and no other markings are reachable from this state, the tree stops at this leaf and we must consider the markings generating by firing t_2 from M_1 (other branch)

$$M_1\,[t_2 > = M_3 = [1,\ 0,\ 1,\ 0]\,,\ t_1 \ \text{Enabled}$$

Notice that M_3 and M_0 have the following property: a common transition enabled, t_1, and a number of tokens on M_3 equal to that of M_0 except for one, $M_3(p_3) > M_0(p_3)$ and, by firing t_1 we will obtain

$$M_3\,[t_1> = M_4 = [0,\ 1,\ 2,\ 0]\,,\quad t_2 \text{ Enabled and } t_3 \text{ Enabled}$$

Notice that M_4 and M_1 share in comoon the same features as M_3 and M_0 before the firing of t_1: common transitions enabled, t_2 and t_3, and a numer of tokens on M_3 equal to that of M_2 except for one, $M_4(p_3) > M_1(p_3)$. If t_3 is fired we will obtain

$$M_4\,[t_3> = M_5 = [0,\ 0,\ 2,\ 1]\,,\quad \text{no transitions Enabled}$$

which is a dead marking *similar* to M_2: both dead, and a number of tokens on M_5 equal to that of M_2 except for one, $M_5(p_3) > M_2(p_3)$. Now, starting from M_4 and by firing t_2

$$M_4\,[t_2> = M_6 = [1,\ 0,\ 2,\ 0]\,,\ t_1 \text{ Enabled}$$

The marking M_6 has the same characteristics as M_3: a common transition enabled, t_1, and a number of tokens on M_3 equal to that of M_0 except for one, $M_6(p_3) > M_3(p_3)$. It is easy to see that place p_3 acts like a token *piggybank* when considering the following transition sequence

$$s = t_1\,t_2\,t_1\,t_2\,\ldots\,t_3$$

and the reachability tree appears endless due to the unbounded feature of the underlying Petri Net (the tokens in the p_3 place are growing without any limit). To overcome this trouble in the characterization of the Reachability set and their related structures (tree/graph) a symbol is added in place of the growing tokens inside p_3, ω which stands for infinite, when their number is greater than or equal to 1. As a consequence the following rewriting on the transitions is performed

$$M_3 = M_6 = [1,\ 0,\ \omega,\ 0]$$
$$M_4 = [0,\ 1,\ \omega,\ 0]$$
$$M_5 = [0,\ 0,\ \omega,\ 1]$$

These three *states* are equivalence classes representing all infinite states whose number of tokens on place p_3 grows without limit. The following tree structure can then be obtained

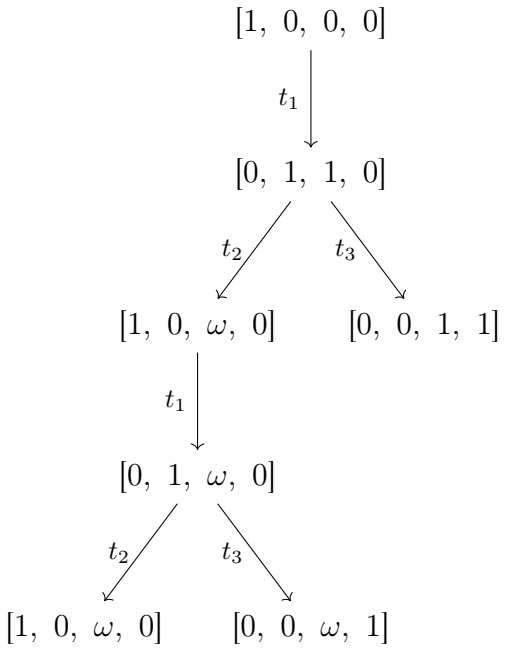

Figure 1.16: Coverability Tree

and is the *Coverability Tree* of Petri Net in Figure 1.15 whose Reachability Set is infinite. The graph structure named *Coverability graph* is as follows

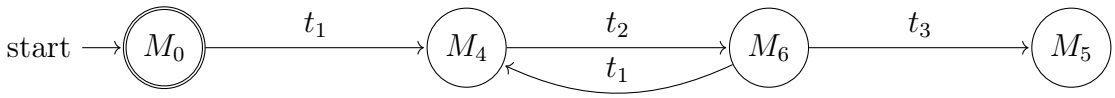

Figure 1.17: Coverability Graph

We We will present an algorithm for constructing a finite coverability tree. To do so, we first introduce some notation:

- *Root node.* This is the first node of the tree, corresponding to the initial state of the given Petri net. For example, $M_0 = [1,\ 0,\ 0,\ 0]$ is the root node in the tree in Fig. 1.16.

- *Terminal node.* This is any node from which no transition can fire (dead marking). For example, in the tree of Fig. 1.16, $[0,\ 0,\ 1,\ 1]$ is a terminal node.

- *Duplicate node.* This is a node that is identical to a node already in the tree. Often, this definition requires that the identical node be in the path from the root to the node under consideration. For example, in the tree of Fig. 1.13, node $[1,\ 1,\ 0]$ resulting from the sequence $t_1\, t_2$ is a duplicate node of the root node.

- *Node dominance.* Let $x = [x(p_1), \ldots, x(p_n)]$ and $y = [y(p_1), \ldots, y(p_n)]$ be two states, i.e., nodes in the coverability tree. We will say that x *dominates* y, denoted by $x >_d y$, if the following two conditions hold:

1. $x(p_i) \geq y(p_i)$, for all $i = 1, \ldots, n$;
2. $x(p_i) > y(p_i)$, for at least some $i = 1, \ldots, n$.

For example, in the tree of Fig. 1.16 we have [1, 0, 2, 0] $>_d$ [1, 0, 1, 0]. But [1, 0, 2, 0] does not dominate [0, 1, 1, 0]. Note that condition 1 above is the definition of coverability for states x, y; however, dominance requires the additional condition 2. Thus, dominance corresponds to *strict* covering.

- *The symbol* ω. This may be thought of as "infinity" in representing the marking (state component) of an unbounded place. We use ω when we identify a node dominance relationship in the coverability tree. In particular, if $x >_d y$, then for all i such that $x(p_i) > y(p_i)$ we replace the value of $x(p_i)$ by ω. Note that adding tokens to a place which is already marked by ω, does not have any effect, that is, $\omega + k = \omega$ for any $k = 0, 1, 2, \ldots$. As an example, in Fig. 1.16 we have [1, 0, 1, 0] $>_d$ [1, 0, 0, 0]. We can then replace [1, 0, 1, 0] by [1, 0, ω, 0].

Coverability Tree construction algorithm

- Step 1: Initialize $x = M_0$ (initial marking).
- Step 2: For each new node, x, evaluate the transition function $x\,[t_j >$ for all enabled transitions $t_j \in T$ in x:
  - Step 2.1: If no transition is enabled at state x, then mark x as a terminal node.
  - Step 2.2: For each enabled transition t_j , create a new node $x\,[t_j > x^+$. If necessary, adjust the marking of node x^+ as follows:
    - Step 2.2.1: If $x(p_i) = \omega$ for some p_i , set $x^+(p_i) = \omega$.
    - Step 2.2.2: If there exists a node y in the path from the root node x_0 (included) to x such that $x^+ >_d y$, set $x^+(p_i) = \omega$ for all p_i such that $x^+(p_i) > y(p_i)$.
    - Step 2.2.3: Otherwise, $x^+(p_i)$ is as obtained in $x\,[t_j >$.

  If node x^+ is identical to a node in the path from x_0 to x, then mark x^+ as a duplicate node.
- Step 3: If all new nodes have been marked as either terminal or duplicate nodes, then stop.

It can be shown that the coverability tree constructed by this algorithm is indeed finite. The proof (which is beyond the scope of this book) is based on a straightforward contradiction argument making use of some elementary properties of trees and of sequences of non-negative integers.

Boundedness, Safety, and Blocking Problems

The problem of boundedness is easily solved using a coverability tree. A necessary and sufficient condition for a Petri net to be bounded is that the symbol ω never appears in its

coverability tree. Since ω represents an infinite number of tokens in some place, if ω appears in place p_i , then p_i is unbounded. For example, in Fig. 1.15, place p_3 is unbounded; this is to be expected, since every time the ordered sequence of firings t_1, t_2 is perfomed, an additional token appears in that place.

If ω never appears in the coverability tree, then we are guaranteed that the state space of the DES we are modeling is finite. The coverability tree is then the reachability tree, since it contains all the reachable states. Analysis of such systems becomes much easier to handle, since we can examine all possible states and transitions between them. For instance, to answer safety questions concerning states (including the existence of dead markings which characterize deadlock and/or blocking states questions), it suffices to examine the finite reachability tree to determine if any of the *illegal states* is reachable.

Finally, note that if ω does not appear in place p_i, then the largest value of $M(p_i)$ for any state encountered in the tree specifies a bound for the number of tokens in p_i . For example, to check if all places are 1-bounded (safe Petri Net) the coverability (reachability) tree of a Petri net must contain states with 0 and 1 as the only place markings.

4.2 Conservation Problems

A different approach in analyzing a Petri Net is related to the analysis of the incidence matrix by putting in light *static* features instead of the Reachability Set which is a *dynamic* portrait of the Net putting in light the mechanism of the underlying DES.

P-invariant

In the context of a Petri Net, a *P-invariant* (or *place invariant*) is a column vector that represents a set of places whose combined token count remains constant across all reachable markings, regardless of the firing of transitions. P-invariants are used to analyze the conservation properties of the Petri Net, such as checking whether certain resources or token amounts are preserved throughout the system's operation.

Definition 4.2. Given a Place-Transition Petri Net $\mathcal{N}$, a *P-invariant* is an integer vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)^T$ such that:

$$M\gamma = M_0\gamma$$

for each $M \in R(\mathcal{N}, M_0)$.

From the Definition and, by considering the updating law of the Petri Net

$$M = M_0 + s\,A$$

where s is a transition sequence driving the Net marking from M_0 to M and A the incidence matrix of the Net, the P-invariant satisfies the following null-space requirement over the integer domain

$$A\,\gamma = 0 \tag{1.10}$$

The previous equation implies that the firing of any enabled transition does not alter the weighted sum of tokens in the set of places defined by γ.

A P-invariant represents a set of places that together have a *conserved* or *invariant* token count, meaning that the total number of tokens across these places remains constant in all reachable markings. This conservation property is a structural feature of the Petri Net and holds true regardless of the specific marking or transition sequence applied.

By exploiting simple Linear Algebra arguments if γ is a P-invariant the $k\,\gamma$, $k \in \mathbb{Z}$ is a P-invariant too and if γ_1, γ_2 are P-invariants of the Net, then $\gamma_1 + \gamma_2$ is another P-invariant.

P-invariants are useful for several purposes in the analysis of Petri Nets:

- **Conservation of Resources**: A P-invariant can represent a set of resources that are conserved in the Petri Net, meaning that the total quantity of these resources does not change regardless of transition firings.
- **Boundedness Analysis**: P-invariants can help determine if certain places in the Petri Net are bounded. If the weighted sum of tokens is constant, it implies that the places involved cannot have an unbounded number of tokens.
- **Deadlock Prevention**: By analyzing P-invariants, it may be possible to identify conditions under which deadlocks or resource shortages are prevented, since the constant sum of tokens can indicate a balanced or cyclical use of resources.
- **Verification of System Properties**: Invariant properties provide a means to verify that certain desirable properties hold within the system, such as ensuring that no tokens are lost or gained in closed-loop systems.

An interesting problem consists in the derivation of the smallest set of P-invariants (possibly finite) of a given Petri Net capable to generate all the possible solutions of 1.10.

Definition 4.3. The *support* of a P-invariant γ, indicated as $\|\gamma\|$ is the collection of places related to non-zero components of γ.

Definition 4.4. A P-invariant γ of a Net is a *mimimum support* P-invariant if its support does not contain the support related to other P-invariants of the same Network.

Definition 4.5. A P-invariant γ of a Net is a *canonical* P-invariant if the greatest common divisor of the non-zero elements is equal to 1.

Positive P-invariants

Positive P-invariants are more interesting than generic P-invariants in Petri Net analysis for the following reasons:

1. **Meaningful Conservation Interpretation**:
   A positive P-invariant (where all entries of the P-invariant vector are non-negative integers) indicates a conserved set of resources, which can represent physical quantities, tasks, or other real-world items. When the components of a P-invariant vector are all positive, they correspond to places in the Petri Net that together hold a conserved total number of tokens. This is useful in scenarios where each place represents a specific type of resource, and the total of those resources must remain constant (e.g., parts in a production system, tasks in a queue).

2. **Ensuring Feasibility in Token Conservation**:
   Positive P-invariants ensure that the conservation rule holds in a practically feasible way: no place in the invariant should have a negative or undefined number of tokens. If an invariant vector has negative or zero entries, it may imply that certain places contribute negatively or not at all to the invariant, which can complicate or invalidate its interpretation as a conservation law.

3. **Boundedness Insight**:
   Positive P-invariants are particularly useful in checking for boundedness of the Petri Net. If a positive P-invariant exists, it indicates that the sum of tokens across a set of places remains constant, which can imply that none of the places involved will become unbounded as long as the invariant holds. This is critical for verifying that the system operates within safe limits without overflowing resources.

4. **Deadlock and Liveness Analysis**:
   Positive P-invariants help ensure that resources are not exhausted and can be reused, which is essential for avoiding deadlocks. Since a positive P-invariant enforces a constant sum of tokens across the involved places, it can indicate cyclic behavior or the balanced circulation of resources, which is useful for maintaining the liveness of the system.

5. **System Design and Validation**:
   In systems design, positive P-invariants make it easier to validate that the Petri Net model accurately reflects intended resource flows and conservations. For example, in a manufacturing process, if each place represents a step that requires a fixed number of tokens (e.g., machines, workers, parts), then a positive P-invariant can validate that these resources are balanced and cycled correctly throughout the process.

6. **Clear Interpretation of Functional Roles**:
   Positive entries in a P-invariant indicate that each place involved has a functional role in the system. Non-positive entries (zero or negative) can suggest places that don't contribute meaningfully to the invariant's conservation property, which may or may not have practical utility in analysis. Thus, positive P-invariants often provide a more direct and interpretable structure, aligning well with functional requirements of resource management or flow control in the Petri Net model.

In summary, positive P-invariants align with physical and logical requirements of most systems modeled by Petri Nets, such as conservation of resources, boundedness, and balanced workflow. This makes them more insightful and applicable than generic P-invariants, which may include zero or negative values that could be harder to interpret or apply in practical scenarios.

In what follows we will restrict our attention only to positive P-invariants.

Definition 4.6. A P-invariants positive generator is the smallest positive P-invariants set γ^k, $1 \leq k \leq q$ such that each P-invariant of the Net can be obtained via linear combinations of γ^k. The elements of this set are the minimal P-invariants.

The following propositions describe a strategy instrumental to derive the P-invariant positive generator.

Proposition 4.1. A P-invariant is minimum i.e. belongs to the P-invariants positive generator iff is canonical and minimum support.

Proposition 4.2. A P-invariants positive generator is finite and unique.

Definition 4.7. A Petri Net is said to be covered by P-invariants if every place belongs to the support of at least one P-invariant.

Definition 4.8. A Petri Net is said to be *conservative* if it is covered by positive P-invariants

$$\forall p \in P,\ \exists \text{ a } P\text{-invariant } \gamma,\ \text{s.t. } p \in \|\gamma\|\,,\ \gamma(p) > 0$$

Proposition 4.3. A conservative Petri Net is bounded.

When considering a conservative Petri Net every place is covered by a positive P-invariant and, as a consequence the number of tokens in each place cannot grow indefinitely. The converse is in general not true, a bounded Petri Net could avoid being conservative.

T-invariants

In a dual fashion of the P-invariants it is possible to define invariant quantities for the transitions (T-invariants). T-invariants descrive possible firing sequences making it possible for the Petri Net to revert to the initial marking M_0. The entries of the T-invariants vector describe the number of times a given transition may fire to reproduce a desired marking.

Definition 4.9. A T-invariant for a Petri Net is a row vector $\beta = (\beta_1,\ \beta_2, \ldots,\ \beta_m)$ such that

$$\beta\, A = 0$$

where A is the incidence matrix of the Net.

By referring to the updating law of the Petri Net

$$M = M_0 + s\, A$$

the T-invariant, whenever exists suggest the number of times a transition must fire to recover the initial marking M_0. It not nonetheless suggests how the transitions must be orderly fired and if the transitions are enabled. The existence of a T-invariant does not automatically imply that M_0 can be recovered.

T-invariants are useful in analyzing the dynamic behaviors of Petri Nets, as they indicate sequences of transitions that can occur repeatedly without changing the overall state. Specific applications include:

- **Identifying Cyclic Behaviors**: T-invariants highlight cyclic behaviors within the Petri Net, revealing sequences of actions that can be performed repeatedly without altering the net's marking.

- **Deadlock Detection**: By examining T-invariants, it is possible to detect if certain transitions are essential for returning the system to its initial state. If a system has no non-trivial T-invariants, it might suggest that the Petri Net could reach a deadlock state.

- **Detecting Repetitive Processes**: T-invariants are particularly useful in systems that require repetitive or cyclic processes, such as manufacturing or communication systems, where certain operations must be repeated indefinitely.

- **System Liveness Analysis**: T-invariants help assess the liveness of a system by showing that certain sequences can be repeated, which may indicate that the system does not become permanently inactive.

Siphon

In the context of a Petri Net, a **siphon** (also called a *deadlock* or *trap complement*) is a specific subset of places that has important implications for the system's behavior, particularly regarding deadlocks and liveness. A siphon represents a set of places from which, once tokens are lost, they cannot be replenished under certain conditions. Understanding siphons is essential for analyzing and ensuring the robustness of systems modeled by Petri Nets.

Definition 4.10. Given a Petri Net a subset of places $S \subseteq P$ is called a *siphon* if:

$$\forall t \in T: \quad \text{if } t \text{ has an output arc to some place in } S,$$

$$\text{then } t \text{ also has an input arc from some place in } S.$$

or

$$\bullet S \subseteq S \bullet$$

In other words, every transition that has an outgoing arc leading to a place in S must also have an incoming arc from a place in S.

Properties and Interpretation of a Siphon

A siphon represents a subset of places in the Petri Net that can lose tokens but cannot regain them once certain conditions are met. Specifically, if all the tokens in a siphon are eventually consumed, it may become permanently unmarked (i.e., have zero tokens) if no new tokens are reintroduced from outside the siphon. This condition has implications for the system's behavior:

- **Potential for Deadlock**:
  A siphon without tokens can lead to a situation where some transitions are permanently disabled, as there are no tokens in the relevant input places. This situation is commonly referred to as a *deadlock*. In such a case, parts of the Petri Net may become unproductive or inactive.

- **Non-Liveness of the System**:
  The presence of an unmarked siphon can cause certain transitions to become unreachable or prevent specific sequences of transitions from being executed. This lack of liveness can indicate that the system is in a state where certain operations or processes can no longer occur.

- **Conservation of Resources**:
  A siphon can be interpreted as a subset of resources that are depleted without replenishment from the outside. When modeling systems with conserved resources (e.g., manufacturing processes, network protocols), siphons highlight areas where resources could be exhausted, leading to potential deadlocks.

Relevance of Siphons in Petri Net Analysis

The identification of siphons is crucial for understanding the robustness and reliability of systems modeled by Petri Nets. Siphons are used to:

- **Detect and Prevent Deadlocks**: By identifying siphons early, it is possible to anticipate potential deadlock situations and adjust the design to ensure that tokens can be replenished or alternate paths are provided to avoid deadlock.

- **Enhance System Liveness**: Ensuring that siphons remain marked helps maintain the liveness of the Petri Net, allowing all transitions to remain potentially fireable.

- **Support Resource Management**: In systems where resources are modeled as tokens, siphons help identify areas where resources may be exhausted, helping designers take preventive measures.

A siphon in a Petri Net is a subset of places that can become permanently unmarked if all tokens are removed, leading to potential deadlocks or non-liveness in the system. Siphons are valuable for analyzing the behavior of the Petri Net, especially for detecting deadlocks, ensuring liveness, and managing resources. Identifying and understanding siphons enables the design of more reliable and efficient systems by anticipating and mitigating conditions that could lead to inactivity or failure.

Traps

In the context of a Petri Net, a *trap* is a specific subset of places that has the property of retaining tokens if it initially contains any. The concept of a trap is closely related to system liveness, as it ensures that certain parts of the Petri Net can continue to function once they have tokens. Understanding traps is important for analyzing system behaviors and ensuring that necessary resources or states remain available.

Definition 4.11. Given a Petri Net, a subset of places $Q \subseteq P$ is called a *trap* if:

$$\forall t \in T: \quad \text{if } t \text{ has an input arc from some place in } Q,$$

$$\text{then } t \text{ also has an output arc to some place in } Q.$$

or

$$Q\bullet \subseteq \bullet Q$$

In other words, every transition that has an incoming arc from a place in Q must also have an outgoing arc to a place in Q.

Properties and Interpretation of a Trap

A trap represents a subset of places in the Petri Net that can retain tokens under certain conditions. Specifically, if any place within the trap is marked (contains tokens), then at least one place in the trap will continue to contain tokens in the future as long as the system operates. This characteristic has several important implications:

- **Ensuring Liveness**:
  Since a trap retains tokens once they enter, it helps to maintain the liveness of the Petri Net by ensuring that certain transitions remain enabled, allowing the system to continue functioning.

- **Modeling Sustained Availability of Resources**:
  A trap can represent resources or conditions that, once available, remain accessible in the system. For example, in manufacturing or workflow systems, traps can model parts or tasks that continuously circulate without complete depletion, ensuring the ongoing availability of resources.

- **Avoiding Dead States**:
  If a trap is marked initially, it ensures that the system can avoid reaching a state where no further progress is possible (dead state). Traps help sustain operations by keeping essential parts of the system active.

Relevance of Traps in Petri Net Analysis

Identifying traps in a Petri Net is essential for understanding the conditions under which the system can maintain its operations. Traps are useful in analyzing and designing Petri Nets for several reasons:

- **Supporting System Liveness**: By ensuring that certain places retain tokens, traps help maintain the system's liveness, allowing transitions to continue firing and preventing total inactivity.

- **Maintaining Resource Availability**: Traps ensure that critical resources or states are always present in the system, making them valuable for models that represent cyclic or repetitive processes.

- **Enhancing Robustness**: Knowing that a trap retains tokens helps design more robust systems that are less prone to failure or deadlock, as essential places remain marked.

A trap in a Petri Net is a subset of places that retains tokens once they enter, as long as at least one of its places is initially marked. Traps are important for maintaining system liveness, avoiding dead states, and ensuring resource availability. Identifying traps enables the design and analysis of resilient systems that can continue functioning without losing critical resources or states.