THÈSE POUR OBTENIR LE GRADE DE DOCTEUR
DE L'UNIVERSITÉ DE MONTPELLIER

En Informatique

École doctorale : Information, Structures, Systèmes

Unité de recherche LIRMM

Vérification d'une méthodologie pour la conception de systèmes numériques critiques

Présenté par Vincent IAMPIETRO
Le Date de la soutenance

Sous la direction de David Delahaye
et David Andreu

Devant le jury composé de

[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]
[Nom Prénom], [Titre], [Labo]	[Statut jury]

UNIVERSITÉ
DE MONTPELLIER

Acknowledgements

The acknowledgments and the people to thank go here, don’t forget to include your project advisor…

Contents

List of Figures

List of Tables

List of Abbreviations

SITPN	**S**ynchronously executed **I**nterpreted **T**ime **P**etri **N**et with priorities
VHDL	**V**ery high speed integrated circuit **H**ardware **D**escription **L**anguage
PCI	**P**lace **C**omponent **I**nstance
TCI	**T**ransition **C**omponent **I**nstance
GPL	**G**eneric **P**rogramming **L**anguage
HDL	**H**ardware **D**escription **L**anguage

For/Dedicated to/To my…

Chapter 1

Implementation of the HILECOP high-level models

In this chapter, we present the input formalism of our transformation function: Synchronously executed Interpreted Time Petri Nets with priorities (SITPNs). For the main part, the formalization of the SITPN structure and semantics is the result of two former Ph.D. theses [4, 5]. However, we contributed to the simplification of the definition of the SITPN structure and its semantics. Moreover, we added complementary definitions for the proof of behavior preservation. Our main contribution to this part lies in the implementation of the SITPN structure and semantics with the Coq proof assistant. This chapter is structured as follows: Section 1.1 is a reminder on the PN formalism and also gives an informal presentation of SITPNs; Section 1.2 lays out the formal definitions of the SITPN structure and semantics; Section 1.3 deals with the implementation of SITPNs with the Coq proof assistant.

1.1 Informal presentation of Synchronously executed Petri nets

Here, fundamentals on the Petri net formalism are outlined, and certain classes of Petri nets are described more precisely. Then, the specificities of the Petri nets used to design the behavior of electronic components in the HILECOP methodology are presented. For more information on the topic of Petri nets, the reader can refer to [2], [6], or [3].

1.1.1 Preliminary notions on Petri nets

Petri nets (PNs), invented by C. A. Petri [7], are used to model a broad range of *dynamic* systems: resource sharing between concurrent processes [2], behavior of agents in multi-agent systems [1], behavior of digital components [8]. A Petri net is a directed graph, composed of two types of node: place nodes (*circles*) and transition nodes (*squares* or *lines*). As shown in Figure 1.1, place nodes usually represent a part of the state of the modelled system, here, the states of two computer processes and a semaphore; transition nodes usually refer to events triggering the system evolution (or state changing). We will see later that there exists a lot of different classes of PNs. Figure 1.1 presents an example of the most simple form of PN, namely, the *place-transition* PN. In this chapter, when no precision is given on the class of PN considered, a PN refers to a *place-transition* PN.

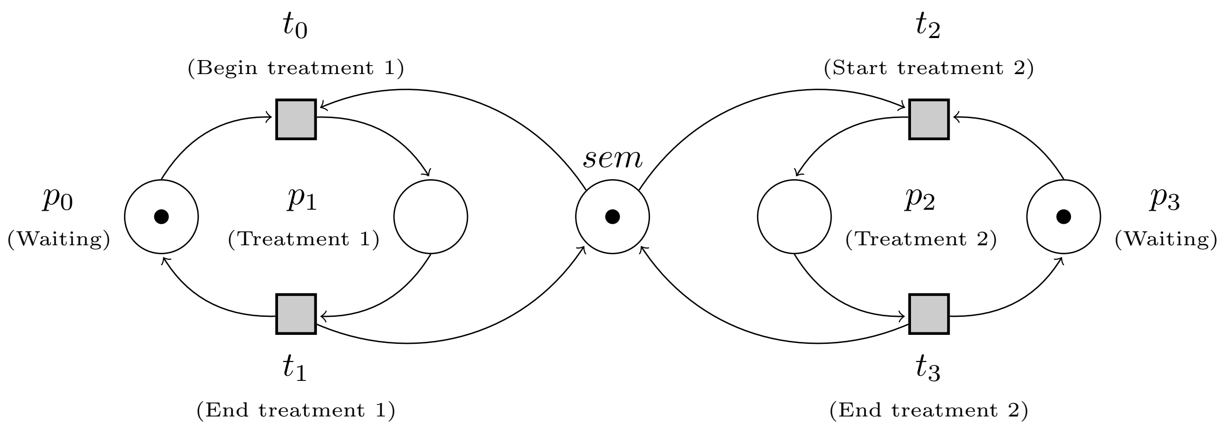


FIGURE 1.1: An example of Petri net. The semaphore place *sem* prevents the parallel execution of *Treatment 1* (place p_1) and *Treatment 2* (place p_2).

Edges

In a Petri net, directed edges link together places and transitions. Places cannot be linked to other places, and the same stands for transitions. There are two kinds of edges, *pre* or *incoming* edges, going from a place to a transition, and *post* or *outcoming* edges, going from a transition to a place. Places linked to a transition t by incoming (resp. outcoming) edges will be referred to as the *input* places (resp. *output* places) of t. The same stands for the transitions linked to a place p. For instance, in Figure 1.1, p_0 and *sem* are the input places of t_0, and p_1 is the output place of t_0; t_1 and t_3 are the input transitions of place *sem*, and t_0 and t_2 are the output transitions of *sem*. Some weight –a natural number– is associated to the edges of a Petri net. If no label appears on the edge then one is the default weight. Petri nets are said to be *generalized* when the weight of the edges are possibly greater than one.

Marking

In Figure 1.1, places p_0, p_3 and *sem* are marked with tokens, represented by little black circles. This means that places p_0, p_3 and *sem* are currently active. The distribution of tokens over places is called the *marking* of the net. The marking of a Petri net reflects the overall state of the modelled system at a certain moment in its activity cycle.

Transition firing

In a Petri net, the marking evolves based on a token consumption-production system. Transitions consume tokens from their input places, and produce tokens to their output places. This whole system is called *transition firing*. In order to be *firable*, a transition must be *sensitized* (or *enabled*), meaning that the number of tokens in each of its input places must be equal or greater than the weight of its incoming edges. For instance, in Figure 1.1, the transition t_0 is sensitized because the weight of the arc (p_0, t_0) is of one (default value), and place p_0 is marked with one token, and the same stands for the number of tokens in place *sem* and the weight of the arc (sem, t_0). As a counter example, transition t_3 is not sensitized because there are no tokens in its input place p_2. Depending on the class of PNs that is considered, other parameters affect the *firability* of transitions (see interpreted Petri nets, time Petri nets and Section 1.1.2). When a sensitized transition is fired, tokens are retrieved from its input places (as many tokens as the weight of the arcs) and produced in its output places (as many tokens as the weight of the arcs). This process represents the occurrence of an event –denoted by the transition– triggering the evolution of the system from one state to another. Figure 1.2 shows the state of the PN of Figure 1.1 after the firing of the transition t_0.

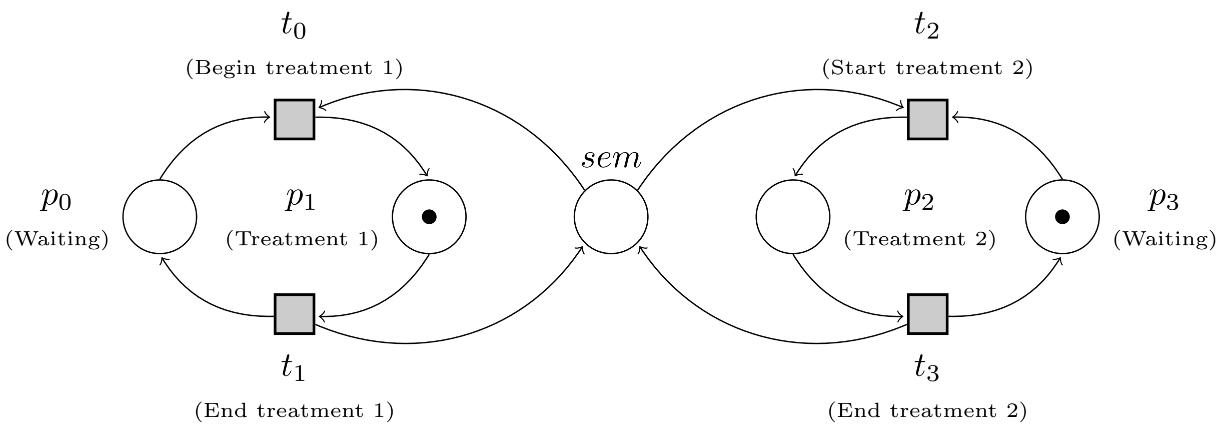


FIGURE 1.2: The PN of Figure 1.1 after the firing of transition t_0.

In Figure 1.2, the tokens in the input places of t_0, i.e. places p_0 and *sem* have been consumed, and one token has been produced in the output place p_1. The current marking indicates that the task "Treatment 1" is being performed (place p_1 is active).

In Figure 1.1, transition t_0 and t_2 are enabled at the same time. However, the *standard* semantics of PNs is such that only one transition can be fired in that case. Either t_0 consumes the token in place *sem* or t_2 does, but never both. Thus, the transition firing process in the standard PN semantics is an nondeterministic process. From the marking of Figure 1.1, two markings are reachable: the marking resulting of the firing of transition t_0 and the one resulting of the firing of transition t_2. Also, the transition firing process is asynchronous. As soon as a transition is enabled, the transition firing process can be triggered.

Extended Petri nets

The class of *extended* Petri nets introduces the inhibitor and test edges. As shown in Figure 1.3, test arc tips are black circles and inhibitor arc tips are white circles. Inhibitor and test edges are incoming edges, always coming from a place toward a transition.

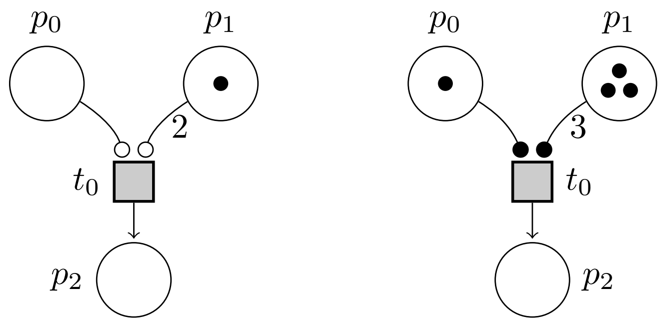


FIGURE 1.3: Two examples of extended Petri nets; on the left side, a PN with inhibitor arcs; on the right side, a PN with test arcs.

The particularity of the inhibitor and test edges is that they are not consuming tokens in input places after the firing of a transition. Indeed, they are just testing the number of tokens in incoming places to determine if the transition is enabled. Inhibitor arcs ensure that the number of tokens in input places is strictly lower than their weights; test arcs ensure that the number of tokens in incoming places is equal or greater than their weights. Therefore, on the left side of Figure 1.3, transition t_0 is sensitized because there is strictly less than one token in place p_0 and strictly less than two tokens in place p_1. On the right side of Figure 1.3, transition t_0 is sensitized because there is at least one token in place p_0 and three tokens in place p_1.

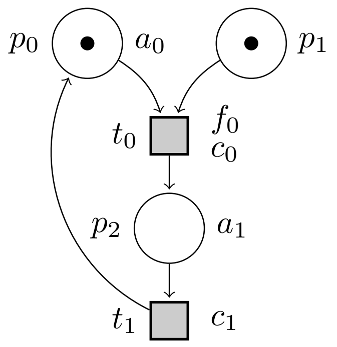
Interpreted Petri nets

Interpreted Petri nets (IPN) [2] describe the interaction between a system and its outside environment. This class of PN introduces three new concepts:

- Continuous actions, associated to the places of a Petri net. Actions associated to a place p are activated as long as p is marked. For instance, when modelling a controller with a IPN, actions can correspond to the setting of a electric signal controlling some actuator (e.g, maintaining a LED on).
- Functions (or discrete actions), associated to the transitions of a Petri net. When a transition t is fired, all functions associated to t are executed. Functions can be any kind of discrete operations –variable incrementation, for instance– manipulating both internal variable and external signal values.
- Conditions, associated to the transitions of a Petri net. Conditions are boolean expressions receiving their values from the environment of the PN. In an IPN, a transition

is firable only if all its associated conditions are `true` (or `false` in the case where an inverse condition is associated).

Conditions represent inputs from the environment; actions and functions describe the influence of the modelled system on its environment. There is an interaction between actions, functions and conditions. The execution of actions and functions modify the environment of the system, and the modification of the environment will be measured through the value of conditions. However, in an IPN, the interaction between actions, functions and conditions is not described as it would require to model the dynamics of the environment. Figure 1.4 illustrates the use of actions, functions and conditions in an interpreted Petri net.



c_0: heat sensor value $> 10°C$
f_0: launch cooler
c_1: heat sensor value $\leq 10°C$
a_0: green LED on
a_1: red LED on

FIGURE 1.4: An example of Interpreted Petri net; on the left side, the interpreted Petri net; on the right side, examples of tests associated to conditions and operations associated to actions and functions.

In Figure 1.4, the action a_0 is activated as place p_0 is marked by one token. Also, function f_0 will be executed at the firing of t_0, that is if condition c_0 is `true` and t_0 is sensitized. On the right side of Figure 1.4, we associate a semantics to conditions, actions and functions in terms of concrete tests or operations. However, when considering the semantics of IPNs, what is of interest to us is the value of conditions and the execution state of actions and functions. We are not interested in interpreting the Boolean expressions associated to conditions but only to retrieve their value. Likewise, we are only interested in the fact that a given action/function is activated/executed but not in what is its effect on the environment.

Time Petri nets

In a time Petri net (TPN), time intervals are associated to transitions. The goal is to constrain the firing of a transition to a certain time window. As shown in Figure 1.5, time intervals are of the form $[a, b]$, where $a \in \mathbb{N}^*$ and $b \in \mathbb{N}^* \sqcup \{\infty\}$. Other definitions of time intervals exist for TPNs (e.g. with real numbers), but here we will only consider the latter definition. In Figure 1.5, time counters are represented in red between diamond brackets. The current value of time counters is part of the state of the TPN, along with its current marking, whereas time intervals are part of the static structure of the TPN.

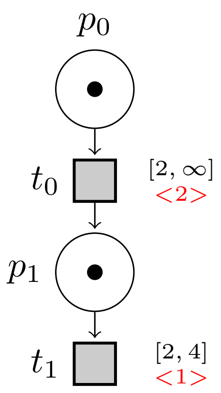


FIGURE 1.5: An example of time Petri net. The value of time counters appears in red.

For each sensitized transition associated with a time interval, time counters are incremented at a certain time step, previously defined by the designer. For instance, in the case of SITPNs, i.e. Petri nets used in the HILECOP methodology, the reference time step for the increment of time counters is the clock cycle.

When a transition associated with a time interval is fired or disabled, a reset order is sent to the transition to set its time counter to zero. The value of reset orders (Boolean values) is also a part of the TPN state. In time Petri nets, a transition is firable only if its time counter value is within its time interval. For instance, in Figure 1.5, only transition t_0 is firable. Moreover, there are several possible firing policies for TPNs. Here, we will only consider the *imperative* firing policy: as soon as a time counter reaches the lower bound of a time interval, the associated transition must be fired.

Petri nets with priorities

Two transitions are in structural conflict if they have a common input place connected through a *basic* arc (i.e. neither inhibitor nor test). When two transitions in structural conflict are firable at the same time and if the firing of one of the transitions disables the other, then, the conflict becomes *effective*. In a Petri net with priorities, it is possible to specify a firing priority in the case where the conflict between two transitions becomes effective. In that case, the transition with the highest firing priority will always be fired first. Figure 1.6 illustrates the application of a priority relation to solve the effective conflict between two transitions.

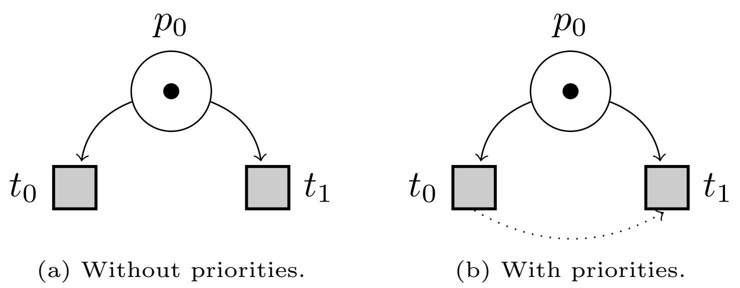


FIGURE 1.6: An example of transitions in structural and effective conflict. In subfigure (b), the dotted arrow represents the priority relation between t_0 and $t1$. The transition with the highest firing priority is at the source of the arrow; here, transition t_0.

1.1.2 Particularities of SITPNs

Here, we will informally present the specificities of the Petri nets describing the internal behavior of the HILECOP high-level model components. These Petri nets are called: Synchronously executed, extended, generalized, Interpreted, Time Petri Nets with priorities or SITPNs. SITPNs are a combination of multiple classes of PNs, namely: extended PNs, generalized PNs, interpreted PNs, time PNs and PNs with priorities. These classes were presented in the above section. We will now talk about another aspect of SITPNs that constitutes the originality of the formalism compared to the standard PN semantics: its synchronous execution.

The class of interpreted Petri nets increases the expressiveness of the HILECOP high-level models. However, to ensure the safe execution of functions after the synthesis of the designed circuit on an FPGA card, the whole system must be synchronized with a clock signal [4]. As a consequence, a clock signal also regulates the evolution of SITPNS (i.e. it is a part of their semantics). The evolution of a SITPN is *synchronized* with two clock events: the rising edge and the falling edge of the signal. Figure 1.7 depicts the process of state evolution, following the clock signal.

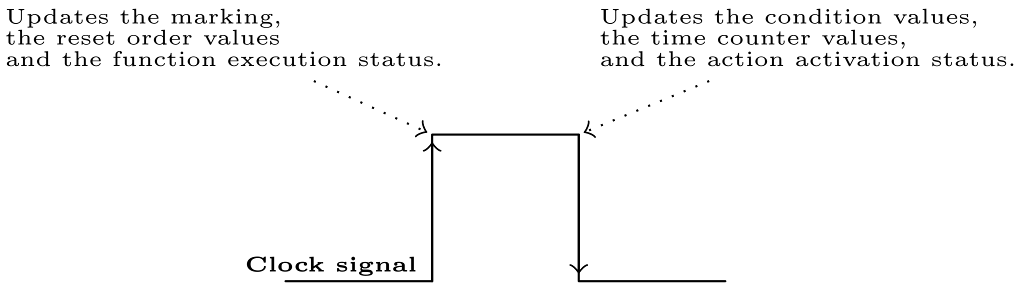

Updates the marking,
the reset order values
and the function execution status.
Updates the condition values,
the time counter values,
and the action activation status.
Clock signal

FIGURE 1.7: Evolution of an SITPN synchronized with a clock signal.

Considering the different classes of PNs that define SITPNs, the state of a SITPN is characterized by its marking, the value of time counters, the reset orders assigned to time counters, the execution/activation status of actions/functions (Boolean values), and the value of conditions (also Boolean). As shown in figure 1.7, the state evolution process of a SITPN is divided into two steps. The rising edge of the clock signal triggers the marking update, which is the consequence of transition firing; all transitions that have been fired or disabled by the firing process receive reset orders; all functions associated with fired transitions are executed. Then, on the falling edge of the clock signal, the environment provides a new value to each condition. Also, the falling edge triggers the evolution of the time counter values; values are incremented, reset, or stalling (see the following remark on locked time counters). Finally, all actions associated with marked places are activated. Figure 1.8 gives an example of the evolution of the state of a given SITPN through one clock cycle. The aim of this figure and the explanation that follows is to give some hints to the reader about the semantics of SITPNs before giving its formal definition in Section 1.2.4.

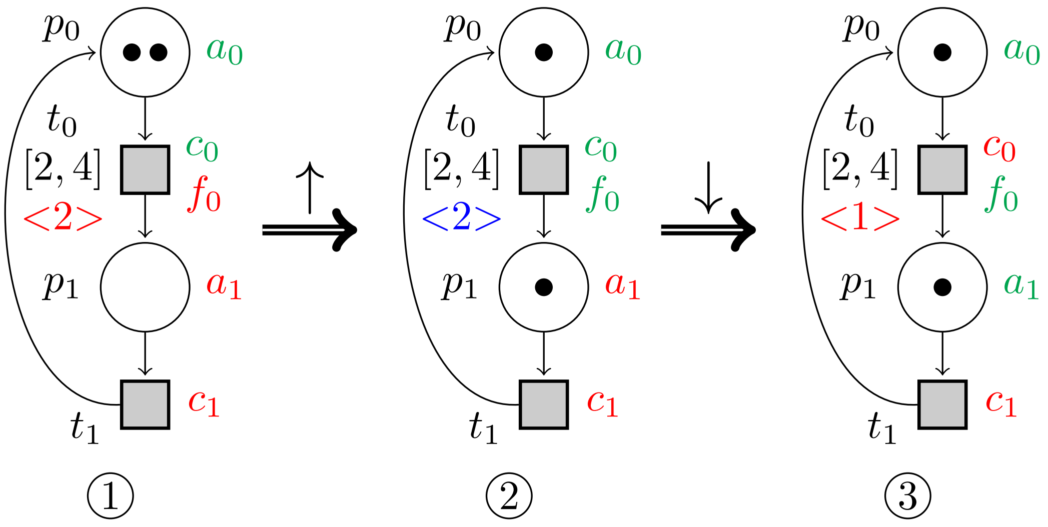


FIGURE 1.8: Evolution of a SITPN over one clock cycle. Conditions appear in green when their value is `true` and in red otherwise; actions and functions appear in green when they are activated/executed and in red otherwise; time counters appear in red and between diamond brackets; time counters appear in blue when they receive reset orders.

From Step 1 to Step 2, the rising edge of the clock signal triggers the SITPN state evolution. Here, transition t_0 is fired. At Step 1, transition t_0 gathers all the necessary conditions to trigger the firing process, namely: t_0 is enabled by the current marking, condition c_0 is `true` (appears in green), the value of t_0's time counter is within the time interval. As a consequence, one token is consumed in place p_0 and one token is produced in place p_1. Also, function f_0 is executed at Step 2 (appears in green) and a reset order is sent to the time counter of t_0 (appears in blue). From Step 2 to Step 3, the falling edge updates the action activation status: a_0 stays activated as place p_0 is still marked; a_1 becomes newly activated as p_1 is marked. The value of time counters are updated: t_0's time counter is set to zero as the transition previously received a reset order. However, as t_0 is still enabled by the new marking, its time counter is incremented. Thus, the resulting time counter value at Step 3 is of one (i.e. result of reset plus increment). Also, the environment provides a new value to each condition. As a consequence, condition c_0 takes the value `false` and condition c_1 keeps the same value.

A remark on priorities

The semantics of synchronous execution is that all transitions are fired at the same time. In Figure 1.9, transitions t_0 and t_1 are both sensitized by place p_0, and consequently are both fired at the same time. The system acts as if two tokens were available in place p_0, one for the firing of t_0 and another for the firing of t_1.

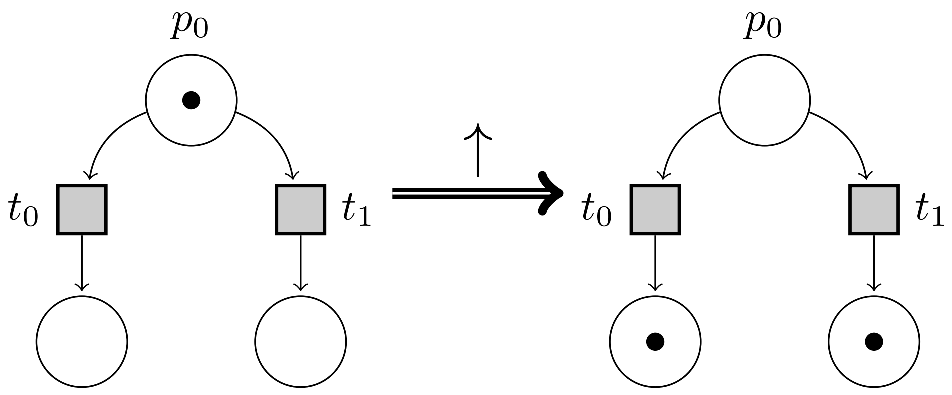


FIGURE 1.9: Double consumption of one token in a SITPN. On the left side, the current marking before the firing of t_0 and t_1; on the right side, the marking resulting of the firing of t_0 and t_1. The arrow indicates the occurrence of a rising edge that triggers the firing process.

In the context of a SITPN, a branching like the one of Figure 1.8, normally interpreted as a disjunctive branching, takes the semantics of a conjunctive branching when no priority are prescribed between the conflicting transitions. To avoid the phenomenon of "double consumption" of tokens, we enforce the resolution of any structural conflict by means of mutual exclusion or through the application of priorities. This policy about the resolution of structural conflicts is part of the definition of a well-defined SITPN presented in Section 1.2.6. The property of well-definition is mandatory to produce safe models of digital systems.

When a structural conflict between transitions is solved with priorities, the firing process follows a slightly different mechanism. As illustrated in Figure 1.10, to determine which transitions of t_0, t_1 and t_2 must be fired, a *residual marking* is computed by following the priority order. For each transition of the group t_0, t_1 and t_2, the residual marking represents the remnant of tokens in p_0 after the firing of transitions with a higher firing priority. Thus, in the semantics of SITPNs, we add an extra condition to the firing of a transition: to be fired, a transition must be enabled by the current marking, must have all its conditions valuated to `true`, must have its time counter within its time interval *and* must be enabled by the residual marking. The computation of the residual marking only involves the consumption phase of the firing process; tokens are withdrawn from places, but none are generated.

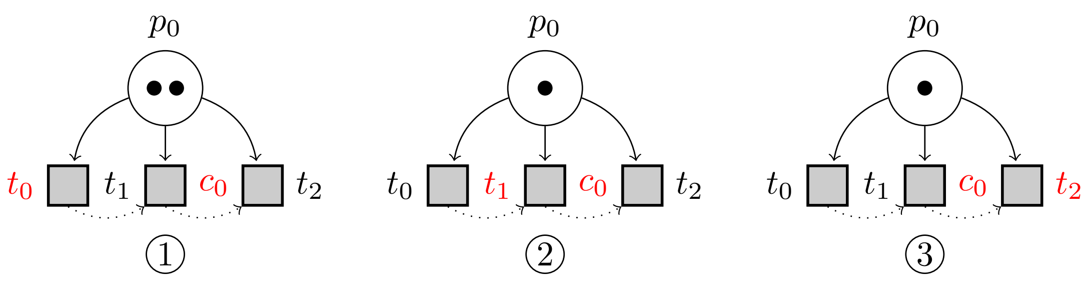


FIGURE 1.10: Computation of the residual marking for a group of conflicting transitions. At ① (resp. ② and ③), the residual marking for transition t_0 (resp. t_1 and t_2). Condition c_0 is in red to indicate that its currentl value is `false`.

In Figure 1.10, the residual marking for t_0 corresponds to the marking obtained after the firing of all transitions with a higher priority. As t_0 is the transition with the highest firing priority, the residual marking for t_0 is equal to the current marking. Transition t_0 gathers all the conditions to be firable and is enabled by the residual marking; thus, t_0 will be fired on the next rising edge. The residual marking for t_1 is the marking obtained after the firing of t_0, i.e. the only transition with a higher priority. As illustrated at ②, t_1 is enabled by the residual marking. However, t_1 does not gather all the conditions to be firable as the value of condition c_0 is `false`. Thus, t_1 will not be fired on the newt rising edge. The residual marking for t_2 is obtained after the firing of t_0 only. Even though transition t_1 has a higher firing priority than t_2, t_1 is not a member of the set of fired transitions. Thus, t_1 is not taken into account in the computation of the residual marking for t_2. The residual marking at ③ enables transition t_2, and as t_2 gathers all the conditions to be firable, then t_2 will be fired on the next rising edge.

Locked time counters

SITPNs inherit the properties of time PNs and interpreted PNs. The phenomenon of *locked* time counters is a consequence of this inheritance. As illustrated in Figure 1.11, the value of a time counter can overreach the upper bound of its associated time interval. This situation can only arise if a condition hinders the firing of a given transition while the considered transition is still enabled by the marking. As a consequence, the time counter will be incremented at every clock cycle until the upper bound of the time interval is overreached. Then, at this point, the time counter is said to be *locked* and its value will no more evolve.

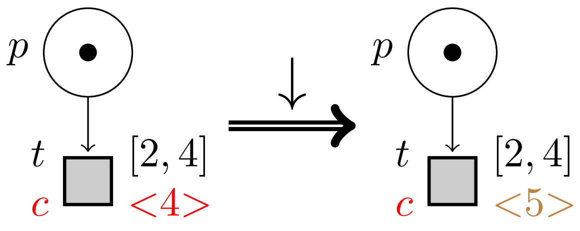


FIGURE 1.11: An example of locked time counter. Condition c is equal to `false` and thus appears in red.

In Figure 1.11, condition c is valuated to `false` before the falling edge of the clock signal. Thus, transition t can not be fired but is still enabled by the marking. On the next falling edge, the time counter of transition t is incremented and overreaches the upper bound of interval $[2, 4]$ and thus becomes locked.

1.2 Formalization of the SITPN structure and semantics

We hope that the reader has now a fair understanding of the concepts underlying the SITPNs and of the dynamics governing the SITPN state evolution process. In this section, we give the formal definition of the SITPN structure and of its execution semantics. We also introduce the concept of a *well-defined* SITPN at the end of the section.

1.2.1 SITPN structure

The structure of SITPNs is formally defined as follows:

Definition 1 (SITPN). *A synchronously executed, extended, generalized, interpreted, and time Petri net with priorities is a tuple* $<P, T, pre, post, M_0, \succ, \mathcal{A}, \mathcal{C}, \mathcal{F}, \mathbb{A}, \mathbb{C}, \mathbb{F}, I_s>$*, where we have:*

1. $P = \{p_0, \dots, p_n\}$*, a finite set of places.*
2. $T = \{t_0, \dots, t_m\}$*, a finite set of transitions.*
3. $pre \in P \to T \nrightarrow (\mathbb{N}^* \times \{\texttt{basic}, \texttt{inhib}, \texttt{test}\})$*, the function associating a weight to place-transition edges.*
4. $post \in T \to P \nrightarrow \mathbb{N}^*$*, the function associating a weight to transition-place edges.*
5. $M_0 \in P \to \mathbb{N}$*, the initial marking of the SITPN.*
6. $\succ \subseteq (T \times T)$*, the priority relation which is a partial order over the set of transitions.*
7. $\mathcal{A} = \{a_0, \dots, a_i\}$*, a finite set of continuous actions.*
8. $\mathcal{F} = \{f_0, \dots, f_k\}$*, a finite set of functions (discrete actions).*

9. *$\mathcal{C} = \{c_0, \ldots, c_j\}$, a finite set of conditions.*

10. *$\mathbb{A} \in P \to \mathcal{A} \to \mathbb{B}$, the function associating actions to places. $\forall p \in P, \forall a \in \mathcal{A}, \mathbb{A}(p, a) =$ `true`, if a is associated to p, $\mathbb{A}(p, a) =$ `false` otherwise.*

11. *$\mathbb{F} \in T \to \mathcal{F} \to \mathbb{B}$, the function associating functions to transitions. $\forall t \in T$, $\forall f \in \mathcal{F}$, $\mathbb{F}(t, f) =$ `true`, if f is associated to t, $\mathbb{F}(t, f) =$ `false` otherwise.*

12. *$\mathbb{C} \in T \to \mathcal{C} \to \{-1, 0, 1\}$, the function associating conditions to transitions. $\forall t \in T$, $\forall c \in \mathcal{C}$, $\mathbb{C}(t, c) = 1$, if c is associated to t, $\mathbb{C}(t, c) = -1$, if $\bar{c}$ is associated to t, $\mathbb{C}(t, c) = 0$ otherwise.*

13. *$I_s \in T \nrightarrow \mathbb{I}^+$, the partial function associating static time intervals to transitions, where $\mathbb{I}^+ \subseteq (\mathbb{N}^* \times (\mathbb{N}^* \sqcup \{\infty\}))$. T_i denotes the definition domain of I_s, i.e. the set of time transitions.*

1.2.2 SITPN State

The SITPN semantics describes the evolution of the state of an SITPN through a given number of clock cycles; thus, we must first define the SITPN state structure:

Definition 2 (SITPN State)**.** *For a given $sitpn \in SITPN$, let $S(sitpn)$ be the set of possible states of $sitpn$. An SITPN state $s \in S(sitpn)$ is a tuple $<M, I, reset_t, ex, cond>$, where:*

1. *$M \in P \to \mathbb{N}$ is the current marking of sitpn.*

2. *$I \in T_i \to \mathbb{N}$ is the function mapping time transitions to their current time counter value.*

3. *$reset_t \in T_i \to \mathbb{B}$ is the function mapping time transitions to time interval reset orders (defined as Booleans).*

4. *$ex \in \mathcal{A} \sqcup \mathcal{F} \to \mathbb{B}$ is the function representing the current activation (resp. execution) state of actions (resp. functions).*

5. *$cond \in \mathcal{C} \to \mathbb{B}$ is the function representing the current value of conditions (defined as Booleans).*

In Items 2 and 3 of Definition 2, *time* transitions refer to transitions with a time interval, i.e. the transitions belonging to the domain of I_s.

1.2.3 Preliminary definitions and fired transitions

Before formalizing the full SITPN semantics, we must introduce some definitions and notations, especially the definition of a *firable* and a *fired* transition. We use the two following notations to simplify the formalization of the SITPN semantics.

Notation 1 (Relations between markings)**.** *For all relation $\mathcal{R}$ existing between two marking functions M and M', the expression $\mathcal{R}(M, M')$ is a notation for $\forall p \in P$, $\mathcal{R}(M(p), M'(p))$. For instance, $M' = M - \sum\limits_{t_i \in Pr(t)} pre(t_i)$ is a notation for $\forall p \in P$, $M'(p) = M(p) - \sum\limits_{t_i \in Pr(t)} pre(p, t_i)$.*

Notation 2 (Sum expressions and arc types)**.** *Many times in this document, we need to express the number of tokens coming in or out of places, after the firing of a certain subset of transitions. To do so, we use two kinds of sum expression:*

1. *The first kind of expression computes a number of output tokens. For instance, for a given place p,* $\sum_{t \in T'} pre(p,t)$ *where* $T' \subseteq T$.

   The expression $\sum_{t \in T'} pre(p,t)$ *is a notation for* $\sum_{t \in T'} \begin{cases} \omega \text{ } if \text{ } pre(p,t) = (\omega, \texttt{basic}) \\ 0 \text{ } otherwise \end{cases}$.

   When computing a sum of output tokens (i.e. resulting of a firing process), we want to add to the sum the weight of the arc between place p and a transition $t \in T'$ *only if there exists an arc of type* `basic` *from p to t (remember that the test and inhibitor never lead to the withdrawal of tokens during the firing process). Otherwise, we add 0 to the sum as it is a neutral element of the addition operator over natural numbers.*

2. *The second kind of expression computes a number of input tokens. For instance, for a given place p,* $\sum_{t \in T'} post(p,t)$ *where* $T' \subseteq T$.

   The expression $\sum_{t \in T'} post(p,t)$ *is a notation for* $\sum_{t \in T'} \begin{cases} \omega \text{ } if \text{ } post(t,p) = \omega \\ 0 \text{ } otherwise \end{cases}$.

   Here, we add the weight of the arc from t to p only if there exists such an arc; we add 0 to the sum otherwise.

Therefore, in the remainder of the document, we will use the conciser notation $\sum_{t \in T'} pre(p,t)$ *to denote an output token sum, and* $\sum_{t \in T'} post(t,p)$ *to denote an input token sum.*

We give the formal definition of the sensitization (see Section 1.1.1 for an informal definition) of a transition by a given marking as follows:

Definition 3 (Sensitization)**.** *A transition* $t \in T$ *is said to be sensitized, or enabled, by a marking M, which is noted* $t \in Sens(M)$*, if* $\forall p \in P, \omega \in \mathbb{N}^*$, $\big(pre(p,t) = (\omega, \texttt{basic}) \vee pre(p,t) = (\omega, \texttt{test})\big) \Rightarrow M(p) \geq \omega$*, and* $pre(p,t) = (\omega, \texttt{inhib}) \Rightarrow M(p) < \omega$.

We give the formal definition of a *firable* transition at a given SITPN state as follows:

Definition 4 (Firability)**.** *A transition* $t \in T$ *is said to be firable at a state* $s = <M, I, reset_t, ex, cond>$*, which is noted* $t \in Firable(s)$*, if* $t \in Sens(M)$*, and* $t \notin T_i$ *or* $I(t) \in I_s(t)$*, and* $\forall c \in \mathcal{C}, \mathbb{C}(t,c) = 1 \Rightarrow cond(c) = 1$ *and* $\mathbb{C}(t,c) = -1 \Rightarrow cond(c) = 0$.

As explain in Section 1.1.2, the firability conditions are not sufficient for a transition to be fired. A transition must also be enabled by the residual marking to go through the firing process. Definition 5 gives the formal definition of a fired transition at a given SITPN state:

Definition 5 (Fired). *A transition $t \in T$ is said to be fired at the SITPN state $s = <M, I, reset_t, ex, cond>$, which is noted $t \in Fired(s)$, if $t \in Firable(s)$ and $t \in Sens(M - \sum_{t_i \in Pr(t)} pre(t_i))$, where $Pr(t) = \{t_i \mid t_i \succ t \land t_i \in Fired(s)\}$.*

One can notice that the definition of the set of fired transitions is recursive. Indeed, to compute the residual marking necessary to the definition of a fired transition, the Pr set must be defined. For a given transition t, the Pr set of t represents all the transitions with a higher firing priority than t that are also fired transitions; hence the recursive definition.

In Definition 5, the marking $M - \sum_{t_i \in Pr(t)} pre(t_i)$ formally qualifies the residual marking for a given transition t and at a given SITPN state s.

1.2.4 SITPN Semantics

We formalize the semantics of a given SITPN as a transition system. The SITPN state transition relation defined in the SITPN semantics as two cases of definition, one for each clock event. The SITPN state transition relation describes the evolution of the state of a SITPN.

Definition 6 (SITPN Semantics). *The semantics of a given $sitpn \in SITPN$ is the transition system $<L, E_c, \rightarrow>$ where:*

- *$s_0 \in S(sitpn)$ is the initial state of the SITPN, such that $s_0 =< M_0, O_{\mathbb{N}}, O_{\mathbb{B}}, O_{\mathbb{B}}, O_{\mathbb{B}} >$, where M_0 is the initial marking of the SITPN, $O_{\mathbb{N}}$ is a function that always returns 0, $O_{\mathbb{B}}$ is a function that always returns `false`.*
- *$L \subseteq \{\uparrow, \downarrow\} \times \mathbb{N}$ is the set of transition labels. A label is a couple (clk, τ) composed of a clock event $clk \in \{\uparrow, \downarrow\}$, and a time value $\tau \in \mathbb{N}$ expressing the current count of clock cycles.*
- *$E_c \in \mathbb{N} \rightarrow \mathcal{C} \rightarrow \mathbb{B}$ is the environment function, which gives (Boolean) values to conditions ($\mathcal{C}$) depending on the count of clock cycles ($\mathbb{N}$).*
- *$\rightarrow \subseteq S(sitpn) \times L \times S(sitpn)$ is the SITPN state transition relation, which is noted $E_c, \tau \vdash s \xrightarrow{clk} s'$ where $s, s' \in S(sitpn)$ and $(clk, \tau) \in L$, and which is defined as follows:*

  □ *$\forall \tau \in \mathbb{N}, \forall s, s' \in S(sitpn)$, we have $E_c, \tau \vdash s \xrightarrow{\downarrow} s'$, where $s =< M, I, reset_t, ex, cond >$ and $s' =< M, I', reset_t, ex', cond' >$, if:*

  (1) $cond'$ is the function giving the (Boolean) values of conditions that are extracted from the environment E_c at the clock count τ, i.e.:
  $\forall c \in \mathcal{C},\ cond'(c) = E_c(\tau, c)$.

  (2) All the actions associated with at least one marked place in the marking M are activated, i.e.:
  $\forall a \in \mathcal{A},\ ex'(a) = \sum_{p \in marked(M)} \mathbb{A}(p, a)$ where $marked(M) = \{p' \in P \mid M(p') > 0\}$.

  (3) All the time transitions that are sensitized by the marking M and received the order to reset their time intervals, have their time counter reset and incremented, i.e.:
  $\forall t \in T_i,\ t \in Sens(M) \land reset_t(t) = \texttt{true} \Rightarrow I'(t) = 1$.

(4) *All the time transitions that are sensitized by the marking M, and did not receive a reset order, increment their time counters if time counters are still active, i.e.:*
$\forall t \in T_i,\ t \in Sens(M) \wedge reset_t(t) = \mathtt{false} \wedge (I(t) \leq upper(I_s(t)) \vee upper(I_s(t)) = \infty) \Rightarrow\ I'(t) = I(t) + 1.$

(5) *All the time transitions verifying the same conditions as above, but with locked counters, keep having locked counters (values are stalling), i.e.:*
$\forall t \in T_i,\ t \in Sens(M) \wedge reset_t(t) = \mathtt{false} \wedge I(t) > upper(I_s(t)) \wedge upper(I_s(t)) \neq \infty \Rightarrow\ I'(t) = I(t).$

(6) *All the time transitions disabled by the marking M have their time counters set to zero, i.e.:*
$\forall t \in T_i,\ t \notin Sens(M) \Rightarrow I'(t) = 0.$

□ $\forall \tau \in \mathbb{N}, \forall s, s' \in S(sitpn)$, *we have* $E_c, \tau \vdash s \xrightarrow{\uparrow} s'$, *where* $s =< M, I, reset_t, ex, cond >$ *and* $s' =< M', I, reset'_t, ex', cond >$, *if:*

(7) M' *is the new marking resulting from the firing of all the transitions contained in* $Fired(s)$, *i.e.:*
$\forall p \in P,\ M'(p) = M(p) - \sum_{t \in Fired(s)} pre(p,t) + \sum_{t \in Fired(s)} post(t,p).$

(8) *A time transition receives a reset order if it is fired at state s, or, if there exists a place p connected to t by a* `basic` *or* `test arc` *and at least one output transition of p is fired and the transient marking of p disables t; no reset order is sent otherwise:*

$$\begin{aligned}
&\forall t \in T_i, t \in Fired(s)\\
&\qquad \vee \big(\exists p \in P, \omega \in \mathbb{N}^*,\ pre(p,t) = (\omega, \mathtt{basic}) \vee pre(p,t) = (\omega, \mathtt{test})\\
&\qquad\quad \wedge \sum_{t_i \in Fired(s)} pre(p, t_i) > 0\\
&\qquad\quad \wedge M(p) - \sum_{t_i \in Fired(s)} pre(p, t_i) < \omega\big) \Rightarrow reset'_t(t) = \mathtt{true},\\
&\qquad \text{and } reset'_t(t) = \mathtt{false} \text{ otherwise.}
\end{aligned}$$

(9) *All functions associated with at least one fired transition are executed, i.e:*
$\forall f \in \mathcal{F},\ ex'(f) = \sum_{t \in Fired(s)} \mathbb{F}(t, f).$

Rules (1) to (6) describe the SITPN state evolution at the falling edge of the clock signal. Rules (1) and (2) pertain to the update of condition values and to the update of the activation status of actions. Rules (3), (4), (5) and (6) focus on the update of time counter values. In Rule (4) of the SITPN semantics, the *active* time counters refer to the time counters that have not yet overreached the upper bound of their associated time interval. Of course, a time counter is always active when the upper bound is infinite. In Rule (5), the *locked* time counters refer to the time counters that have overreached the upper bound of their associated time interval. Of course, time counters can never be locked in the presence of an infinite upper bound. In Rules (4) and (5), for a given time interval i, $upper(i)$ denotes the upper bound of the time interval, and $lower(i)$ denotes the lower bound of the time interval.

Rules (7) to (9) describe the SITPN state evolution at the rising edge of the clock signal. Rule (7) corresponds to the marking update. The computation of the new marking uses the set of fired transitions at state s, i.e. $Fired(s)$. Rule (9) pertains to the update of the function execution status. Rule (8) computes the reset orders for time transitions. There are two cases where a time transition receives the order to reset its time counter. First, if the transition is one of the fired transitions at state s, then its time counter must be reset on the next falling edge. Second, if the transition is disabled in a *transient* manner, then its time counter must also be reset. Figure 1.12 illustrates the case of a transition disabled by the *transient* marking, i.e. the marking obtained after the consumption phase of the firing process.

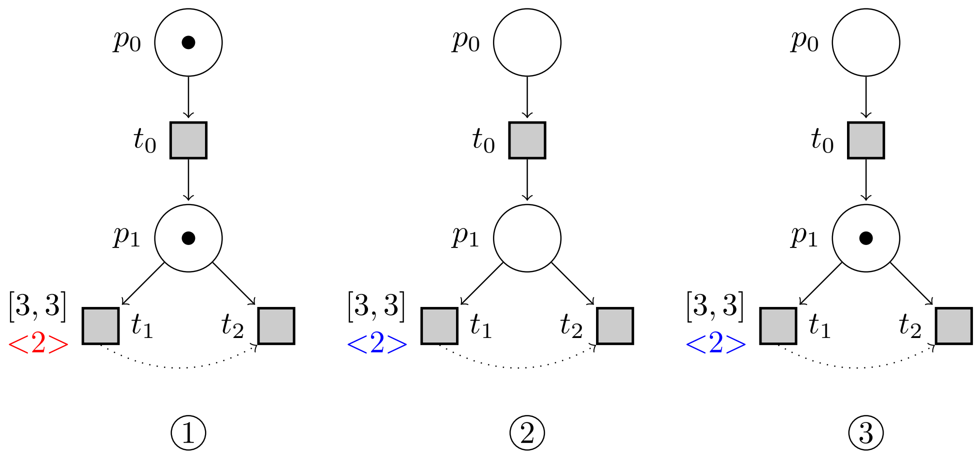


FIGURE 1.12: An example of transition that receives a reset order after being disabled by the transient marking. At ①, the marking before the firing of transitions t_0 and t_2; at ②, the transient marking; at ③, the marking at the end of the firing process.

In Figure 1.12, the situation at ① describes the state of the SITPN before a rising edge. Judging by the current SITPN state at ①, transition t_0 and t_2 will be fired on the next clock event. Situation ② depicts the marking obtained after the consumption phase of the firing process, i.e. the so-called *transient* marking. Situation ③ corresponds to the marking at the end of the firing process. At ③, transition t_1 is enabled by the marking. However, at ②, the transient marking disables t_1 and thus t_1 must receive a reset order (represented by a blue time counter).

1.2.5 SITPN Execution

As a part of the SITPN semantics, we define here the SITPN execution and SITPN full execution relations. These relations bind a given SITPN to the execution trace, i.e. a time-ordered list of states, that it produces when executed over a given number of clock cycles.

These definitions are additional elements corresponding to our own contribution to the formalization of the SITPN semantics.

Definition 7 (SITPN execution). *For a given $sitpn \in SITPN$, a starting state $s \in S(sitpn)$, a clock cycle count $\tau \in \mathbb{N}$, and an environment $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, sitpn yields the execution trace θ from starting state s, written $E_c, \tau \vdash sitpn, s \to \theta$, by following the two rules below:*

EXECUTIONEND

$$\frac{}{E_c, 0 \vdash sitpn, s \to [\,]}$$

EXECUTIONLOOP

$$\frac{E_c, \tau \vdash sitpn, s \xrightarrow{\uparrow} s' \qquad E_c, \tau \vdash sitpn, s' \xrightarrow{\downarrow} s'' \qquad E_c, \tau - 1 \vdash sitpn, s'' \to \theta}{E_c, \tau \vdash sitpn, s \to (s' :: s'' :: \theta)}\ \tau > 0$$

Definition 8 (SITPN full execution). *For a given $sitpn \in SITPN$, a clock cycle count $\tau \in \mathbb{N}$, and an environment $E_c \in \mathbb{N} \to \mathcal{C} \to \mathbb{B}$, sitpn yields the execution trace θ starting from its initial state $s_0 \in S(sitpn)$ (as defined in Definition 6), written $E_c, \tau \vdash sitpn \to \theta$, by following the two rules below:*

FULLEXEC0

$$\frac{}{E_c, 0 \vdash sitpn \xrightarrow{full} [s_0]}$$

FULLEXECCONS

$$\frac{E_c, \tau \vdash s_0 \xrightarrow{\uparrow_0} s_0 \qquad E_c, \tau \vdash s_0 \xrightarrow{\downarrow} s \qquad E_c, \tau - 1 \vdash sitpn, s \to \theta_s}{E_c, \tau \vdash sitpn \xrightarrow{full} (s_0 :: s_0 :: s :: \theta_s)}\ \tau > 0$$

The FULLEXECCONS rule of the SITPN full execution relation (Definition 8) appeals to the SITPN execution relation (Definition 7). However, the definition of the SITPN full execution relation is necessary because the first cycle of execution, starting from the initial state s_0, is particular. As shown in the premises of Rule FULLEXECCONS, the first rising edge is idle. We consider that no transitions are fired during the first rising edge. Thus, the first rising edge does not change the initial state s_0, and we denote the particular first rising edge with the sign $\uparrow_0$ over the SITPN transition relation.

1.2.6 Well-definition of a SITPN

To be able to transform a given SITPN into a VHDL design and also to perform the proof of semantic preservation, a SITPN must verify some properties ensuring its *well-definition*. Here, we formalize the predicate stating that a given SITPN is well-defined.

The main interest of the well-definition predicate is to prevent the phenomenon of the "double consumption" of tokens at the execution of a SITPN. In a well-defined SITPN, a conflict resolution strategy must be applied to every group of transitions in structural conflict. We must be able to decide which transition in a conflicting pair will be fired when the conflict becomes effective. Thus, we give the formal definition of a conflicting pair of transitions and of a conflict group.

Definition 9 (Conflict)**.** *For a given sitpn* $\in SITPN$*, two transitions* $t, t' \in T$ *are in conflict if there exists a place* $p \in P$ *such that* $p \in input(t) \cap input(t')$ *and there exist* $\omega, \omega' \in \mathbb{N}^*$ *such that* $pre(p,t) = (\omega, \texttt{basic})$ *and* $pre(p,t') = (\omega', \texttt{basic})$*.*

A conflict group qualifies a finite set of transitions that are all in conflict with each other through the same place. In Figure 1.13, the set $\{t_0, t_1\}$ is a conflict group. The formal definition of a conflict group is as follows:

Definition 10 (Conflict Group)**.** *For a given sitpn* $\in SITPN$*,* $T_c \subseteq T$ *is a conflict group if there exists a place p such that* $\forall t \in T, \left(\exists \omega \in \mathbb{N}^*,\ pre(p,t) = (\omega, \texttt{basic})\right) \Leftrightarrow t \in T_c$*.*

Contrary to the statement made in [4, p. 67], we no more consider the notion of conflict as being transitive. To illustrate this, Figure 1.13 shows two conflict groups: $\{t_0, t_1\}$ and $\{t_1, t_2\}$. In a well-defined $SITPN$ (see Section 1.2.6), all conflicts in a conflict group must be dealt with, i.e. for all pair of transitions in the group the conflict must be solved. However, we no more consider transitions t_0 and t_2 as in conflict. We argue that even when no conflict resolution technique is applied between transitions in the same situation as t_0 and t_2, the execution of the $SITPN$ can neither result in the double-consumption of a token, nor in the case where a transition is not elected to be fired even though it ought to be. Therefore, we no more consider the construction of merged conflict group (i.e, conflict groups must be merged into one if their intersection is not empty; e.g, $\{t_0, t_1, t_2\}$ in Figure 1.13) as being necessary.

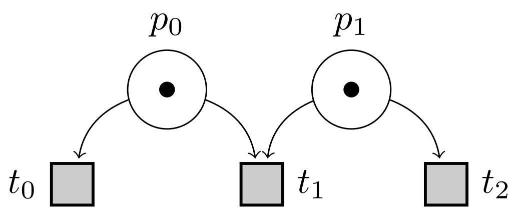


FIGURE 1.13: An example of two separate conflict groups, namely: $\{t_0, t_1\}$ and $\{t_1, t_2\}$.

When the conflict between a pair of transitions becomes effective, there are two ways to be sure that only one transition will be fired. The first way is to define a firing order through a priority relation. The second way is to use a mean of mutual exclusion. A mean of mutual exclusion ensures that the two transitions of a conflicting pair will never be firable at the same time. For now, we only consider three ways of mutual exclusion, namely: mutual exclusion with complementary conditions, mutual exclusion with disjoint time intervals and mutual exclusion with inhibitor arcs. Here, we give the formal definition of these three means of mutual exclusion.

Definition 11 (Mutual exclusion with disjoint time intervals). *Given two conflicting transitions t_0 and t_1, t_0 and t_1 are in mutual exclusion with disjoint time intervals if there exists $a, b \in \mathbb{N}^*$ and $c, d \in \mathbb{N}^* \sqcup \{\infty\}$ such that $I_s(t_0) = [a, b]$ and $I_s(t_1) = [c, d]$ and there is no overlapping between $[a, b]$ and $[c, d]$.*

Definition 12 (Mutual exclusion with complementary conditions). *Given two conflicting transitions t_0 and t_1, t_0 and t_1 are in mutual exclusion with complementary conditions if there exists $c \in \mathcal{C}$ such that $(\mathbb{C}(t_0, c) = 1 \wedge \mathbb{C}(t_1, c) = -1)$ or $(\mathbb{C}(t_0, c) = -1 \wedge \mathbb{C}(t_1, c) = 1)$.*

Definition 13 (Mutual exclusion with an inhibitor arc). *Given two conflicting transitions t_0 and t_1, t_0 and t_1 are in mutual exclusion with an inhibitor arc if there exists $p \in P$ and $\omega \in \mathbb{N}^*$ such that $(pre(p, t_0) = (\omega, \texttt{basic}) \vee pre(p, t_0) = (\omega, \texttt{test})) \wedge pre(p, t_1) = (\omega, \texttt{inhib})$ or $(pre(p, t_1) = (\omega, \texttt{basic}) \vee pre(p, t_1) = (\omega, \texttt{test})) \wedge pre(p, t_0) = (\omega, \texttt{inhib})$.*

Figure 1.14 illustrates the three means of mutual exclusion that can be applied to solve a conflict between two transitions.

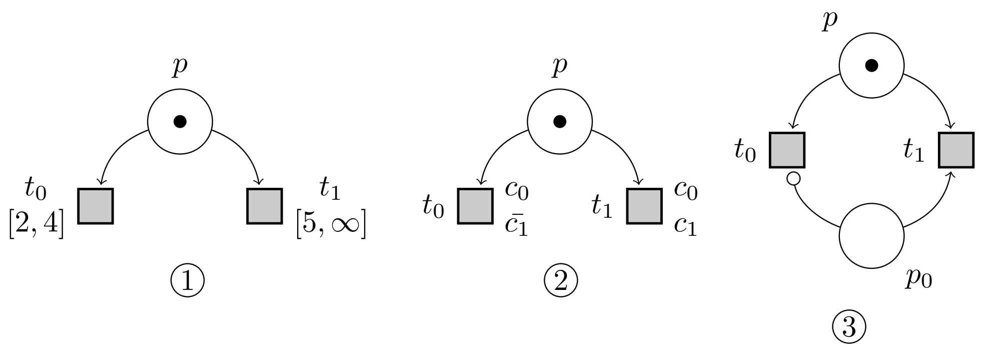


FIGURE 1.14: Examples of conflicting transitions in mutual exclusion. At ①, an example of mutual exclusion with disjoint time intervals; at ②, an example of mutual exclusion with complementary conditions; at ③, an example of mutual exclusion with an inhibitor arc.

In Figure 1.14, in situation ②, condition c_1 is associated to t_1 and the complementary condition is associated to t_0 thus creating the mutual exclusion. In situation ③, the arc (p_0, t_0) and (p_0, t_1) ensure the mutual exclusion between transitions t_0 and t_1. Note that in the structure of mutual exlcusion with an inhibitor arc, the weight of the inhibitor arc and of the basic or test arc must be the same; otherwise, the mutual exclusion is not effective.

A given $sitpn \in SITPN$ is well-defined if it enforces some properties needed on the HILECOP source models before the transformation into VHDL. If the properties, layed out in Definition 14, are not ensured, they will lead to compile-time errors during the transformation of the SITPN into a VHDL design.

Definition 14 (Well-defined SITPN). *A given $sitpn \in SITPN$ is well-defined if:*

- *$T \neq \emptyset$, the set of transitions must not be empty.*
- *$P \neq \emptyset$, the set of places must not be empty.*

- *There is no isolated place, i.e, a place that has neither input nor output transitions:* $\nexists p \in P,\ input(p) = \emptyset \wedge output(p) = \emptyset$*, where* $input(p)$ *(resp.* $output(p)$*) denotes the set of input (resp. output) transitions of* p*.*
- *There is no isolated transition, i.e, a transition that has neither input nor output places:* $\nexists t \in T,\ input(t) = \emptyset \wedge output(t) = \emptyset$*, where* $input(t)$ *(resp.* $output(t)$*) denotes the set of input (resp. output) places of* t*.*
- *For all conflict group as defined in Definition 10, either all conflicts (i.e. for all pair of transitions in the conflict group) are solved by one of the mean of mutual exclusion, or, the priority relation is a strict total order over the transitions of the conflict group.*

1.3 Implementation of the SITPN structure and semantics

In this section, we present our mechanization of the SITPN structure and semantics with the Coq proof assistant. The source code is available to the reader at the address `https://github.com/viampietro/ver-hilecop`. More precisey, the implementation of the SITPN structure and semantics is to be found under the `sitpn/dp` directory. We have made a first implementation of SITPNs without the use of dependent types. For this first version, we have also implemented a SITPN interpret (a so-called *token player*) and proved that the interpret was sound and complete w.r.t the SITPN semantics. This first implementation of the SITPNs and the formal proof of soundness and completeness are available at `https://github.com/viampietro/sitpns`.

1.3.1 Implementation of the SITPN and the SITPN state structure

Listing 1.1 presents the implementation of the SITPN structure as a Coq record type. The implementation is almost similar to the formal definition of the SITPN structure given in Definition 1.

```
Record Sitpn := BuildSitpn {

  places : list nat;
  transitions : list nat;
  P := { p | (fun p0 ⇒ In p0 places) p };
  T := { t | (fun t0 ⇒ In t0 transitions) t };

  pre : P → T → option (ArcT * ℕ*);
  post : T → P → option ℕ*;
  M₀ : P → nat;
  Iₛ : T → option TimeInterval;

  conditions : list nat;
  actions : list nat;
  functions : list nat;
  𝒞 := { c | (fun c0 ⇒ In c0 conditions) c };
17     𝒜 := { a | (fun a0 ⇒ In a0 actions) a };
18     ℱ := { f | (fun f0 ⇒ In f0 functions) f };
19
20     ℂ : T → 𝒞 → MOneZeroOne;
21     𝔸 : P → 𝒜 → bool;
22     𝔽 : T → ℱ → bool;
23
24     pr :  T → T → Prop;
25
26   }.
```

LISTING 1.1: Implementation of the SITPN structure in Coq.

We use lists of natural numbers, i.e. `list nat` in Coq, to define the finite sets of places (Line 3), transitions (Line 4), actions (Line 14), conditions (Line 13) and functions (Line 15) in the `Sitpn` record. We want to use these finite sets in the signature of functions appearing in the structure (e.g use the finite set of places P in the signature of the initial marking $M_0 \in P \rightarrow \mathbb{N}$). To do son we leverage the Coq `sig` type to define subsets of elements verifying a certain property. Thus, we define the finite set P as the subset of natural numbers that are members of the `places` list (Line 5). We use the `In` relation defined in the Coq standard library to express the membership of a natural number regarding the elements of `places` list. Also, the `ArcT` type (Line 8) implements the set $\{\texttt{inhib}, \texttt{test}, \texttt{basic}\}$; the `TimeInterval` type (Line 11) implements the set $\mathbb{I}^+$ of time intervals, and the `MOneZeroOne` type (Line 20) implements the set $\{0, 1, -1\}$. The priority relation is implemented by the `pr` function (Line 24) taking two transitions in parameter and projecting to the type of logical propositions, i.e. the `Prop` type.

Listing 1.2 presents the implementation of the SITPN state structure as a Coq record type.

```
Record SitpnState (sitpn : Sitpn) := BuildSitpnState {

  M :  P sitpn → nat;
  I :  Tᵢ sitpn → nat;
  reset :  Tᵢ sitpn → bool;
  cond :  C sitpn → bool;
  ex :  A sitpn + F sitpn → bool;

}.
```

LISTING 1.2: Implementation of the SITPN state structure in Coq.

The `SitpnState` type definition depends on a given SITPN passed as a parameter; it is an example of dependent type. Projection functions are automatically generated to access the attributes of a record at the declaration of a type with the `Record` keyword. Thus, in Listing 1.2, we can refer to the set of places of `sitpn` with the term `P sitpn`. The term T_i `sitpn` denotes the set of time transitions of `sitpn`. The set of time transitions for a given SITPN is declared as a `sig` type qualifying to the subset of transitions with an associated time interval.

1.3.2 Implementation of the SITPN semantics

Here, we present our implementation of the SITPN semantics. In Listing 1.3, we give an excerpt of the implementation of the SITPN state transition relation, i.e. the core of the SITPN semantics.

```
Inductive SitpnStateTransition
  (sitpn : Sitpn) (Eс : nat → C sitpn → bool) (τ : nat) (s s' : SitpnState sitpn) :
  Clk → Prop :=
| SitpnStateTransition_falling :

    (* Rule (2) *)
    (forall a marked sum,
       Sig_in_List (P sitpn) (fun p ⇒ M s p > 0) marked →
       BSum (fun p ⇒ 𝔸 p a) marked sum →
       ex s' (inl a) = sum) →

    (* Rules (3), (4), (5) and (6) *)
    (forall (t : Ti sitpn), ~Sens (M s) t → I s' t = 0) →
    (forall (t : Ti sitpn), Sens (M s) t → reset s t = true → I s' t = 1) →
    (forall (t : Ti sitpn),
        Sens (M s) t →
        reset s t = false →
        (TcLeUpper s t ∨ upper t = i+) → I s' t = S (I s t)) →
    (forall (t : Ti sitpn),
        Sens (M s) t →
        reset s t = false →
        (upper t <> i+ ∧ TcGtUpper s t) → I s' t = S (I s t)) →

    (** Conclusion *)
    SitpnStateTransition Eс τ s s' ↓

| SitpnStateTransition_rising:

    (** Rule (7) *)
    (forall fired, IsNewMarking s fired (M s')) →

    (* Rule (9) *)
    (forall f fired sum,
       IsFiredList s fired →
       BSum (fun t ⇒ 𝔽 t f) fired sum →
       ex s' (inr f) = sum) →

    (* Conclusion *)
    SitpnStateTransition Eс τ s s' ↑.
```

LISTING 1.3: Excerpt of the implementation of the SITPN state transition relation in Coq.

The SITPN state transition relation is implemented in Coq as an inductive type with two constructors, i.e. one for each clock event. The relation has 6 parameters: an SITPN, an environment E_c, a clock count τ, two SITPN states `s` and `s'` and a clock event. Note that the two states `s` and `s'` are bound to the SITPN parameter through their type, i.e. `SitpnState sitpn`.

In the construction case `SitpnStateTransition_falling`, we give the implementation of Rules (2), (3), (4), (5) and (6) defined in the SITPN semantics. The sum term of Rule (2), i.e. $\sum_{p \in marked(M)} \mathbb{A}(p,a)$, is implemented by Lines 8 and 9. At Line 8, the `Sig_in_List` predicate states that all the inhabitant of the `P sitpn` type (i.e. the places of `sitpn`) that verifies the property `(fun p => M s p > 0)` (i.e. the marking of a place is greater than zero at state `s`) are members of the `marked` list. Because we can not iterate over the elements of a given `sig` type, we use the `Sig_in_List` relation to convert a `sig` type into a list. Lists are iterable by definition. At Line 9, the `BSum` relation states that `sum` is the Boolean sum obtained by applying the function `(fun p => 𝔸 p a)` to the elements of the `marked` list. Rules (3), (4), (5) and (6) are almost similar in their implementation to the description of Definition 6. The Coq term `Sens (M s) t` implements the term $t \in Sens(M)$. Due to the particular nature of the upper bound of a time interval, i.e. defined over the set $\mathbb{N}^* \sqcup \{\infty\}$, the test that the current time counter of a given transition t is less than or equal to the upper bound is implemented by a separate predicate `TcLeUpper`. Similarly, the `TcGtUpper` predicate implements the inverse test.

In the construction case `SitpnStateTransition_rising`, we give the implementation of Rules (7) and (9) defined in the SITPN semantics. In the implementation of Rule (7), the `IsNewMarking` predicate hides away the expression $\forall p \in P,\ M'(p) = M(p) - \sum_{t \in Fired(s)} pre(p,t) + \sum_{t \in Fired(s)} post(t,p)$. In its definition, the `IsNewMarking` predicate first checks that the `fired` list implements the set of fired transitions at state `s`. Then, it builds the marking at state `s'` for each place p, i.e. `(M s')`, by consuming and producing a number of tokens starting from the marking of p at state `s`. The `fired` list is helpful to qualify the input token sum and the output token sum for a given place. Similarly to the implementation of Rule (2), the implementation of Rule (9) at Line 33 leverages the `BSum` predicate to compute the Boolean sum $\sum_{t \in Fired(s)} \mathbb{F}(t,f)$. The term `IsFiredList s fired` states that the `fired` list implements the set of fired transitions at state `s` so we can use the `fired` list to compute the above sum.

Bibliography

[1] Jose R. Celaya, Alan A. Desrochers, and Robert J. Graves. “Modeling and Analysis of Multi-Agent Systems Using Petri Nets”. In: *2007 IEEE International Conference on Systems, Man and Cybernetics*. 2007 IEEE International Conference on Systems, Man and Cybernetics. Oct. 2007, pp. 1439–1444. DOI: 10.1109/ICSMC.2007.4413960.

[2] René David and Hassane Alla. “Petri Nets for Modeling of Dynamic Systems: A Survey”. In: *Automatica* 30.2 (Feb. 1, 1994), pp. 175–202. ISSN: 0005-1098. DOI: 10.1016/0005-1098(94)90024-8. URL: https://www.sciencedirect.com/science/article/pii/0005109894900248 (visited on 06/17/2021).

[3] Michel Diaz. *Les Réseaux de Petri: Modèles Fondamentaux*. Hermès science publications, 2001.

[4] Hélène Leroux. “Méthodologie de conception d’architectures numériques complexes : du formalisme à l’implémentation en passant par l’analyse, préservation de la conformité. Application aux neuroprothèses”. PhD thesis. Université Montpellier II - Sciences et Techniques du Languedoc, Oct. 28, 2014. URL: https://tel.archives-ouvertes.fr/tel-01766458 (visited on 02/10/2020).

[5] Ibrahim Merzoug. “Validation formelle des systèmes numériques critiques : génération de l’espace d’états de réseaux de Petri exécutés en synchrone”. PhD thesis. Université Montpellier, Jan. 15, 2018. URL: https://tel.archives-ouvertes.fr/tel-01704776 (visited on 02/10/2020).

[6] T. Murata. “Petri Nets: Properties, Analysis and Applications”. In: *Proceedings of the IEEE* 77.4 (Apr. 1989), pp. 541–580. ISSN: 1558-2256. DOI: 10.1109/5.24143.

[7] Carl Adam Petri. “Kommunikation mit Automaten”. In: *http://edoc.sub.uni-hamburg.de/informatik petri.pdf* (1962). URL: https://edoc.sub.uni-hamburg.de//informatik/volltexte/2011/160/ (visited on 06/10/2021).

[8] Alex Yakovlev and Albert Koelmans. “Petri Nets and Digital Hardware Design”. In: *Lecture Notes in Computer Science - LNCS*. Apr. 11, 2006, pp. 154–236. DOI: 10.1007/3-540-65307-4_49.