Adding Real Time Constraints to Synchronised Petri Nets

Giovanna Di Marzo Serugendo[1], Dino Mandrioli[2]*,
Didier Buchs[3], Nicolas Guelfi[4]

[1] CERN/IT Division, CH-1211 Geneva 23, Switzerland

[2] Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano 20133, Italy

[3] LGL-DI, Swiss Federal Institute of Technology, CH-1015 Lausanne, Switzerland

[4] Department of Applied Computer Science,

IST - Luxembourg University of Applied Science, L-1359 Luxembourg-Kirchberg

Giovanna.Di.Marzo@cern.ch

October 27, 2000

Abstract

This report defines synchronised Petri nets with inhibitor arcs and an extension of these nets that integrates real-time constraints. The semantics of these nets is given by a transition system built using Structured Operational Semantics (SOS)) rules. This report is part of a larger framework that attempts to attach real-time constraints to the CO-OPN/2 language.
Keywords: CO-OPN, Petri nets, real time, linear temporal logic.

1 Introduction - Motivation

CO-OPN/2 [2] is an object-oriented specifications formalism based on algebraic data types [7] (ADT) and Petri nets which are combined in a way that is similar to algebraic nets [6]. Algebraic specifications are used to describe the data structures and the functional aspects of a system, while Petri nets allow to model the system's concurrent features. To compensate for algebraic Petri nets' lack of structuring capabilities, CO-OPN/2 provides a structuring mechanism based on a synchronous interaction between algebraic nets, as well as notions specific to object-orientation such as the notions of class, inheritance, and sub-typing.

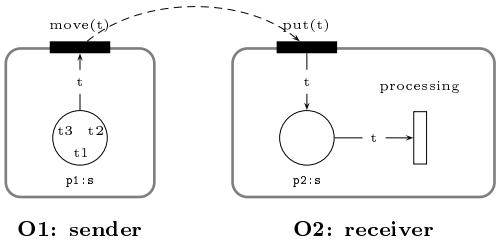


Figure 1: Toy Example with CO-OPN/2

Figure 1 depicts a CO-OPN/2 specification made of two objects (two Petri nets) called **O1** and **O2**. Object **O1** is an instance of Class module `Sender`; it is of type `sender`. It contains a place `p1`, storing terms of sort `s`, and a transition (called method in the CO-OPN/2 context) `move(t)`. The current marking of this place contains the three algebraic terms `t1`, `t2`, `t3`. Object **O2** is an instance of Class `Receiver`; it is of type `receiver`. It contains a place `p2`, storing terms of sort `s`, and a method `put(t)`. Place `p2` is initially empty. Object **O2** contains as well an internal transition (called transition in the CO-OPN/2 context) `processing`. The dashed arrow

*this work has been realized while Dino Mandrioli was visiting the Swiss Federal Institute of Technology.

from method `move(t)` to method `put(t)` stands for the fact that method `move(t)` requests a *synchronisation* with method `put(t)` whenever it fires[1].

The CO-OPN/2 semantics states that : (1) method `move(t)` can be fired only if method `put(t)` is firable; (2) if `move(t)` is fired, then `put(t)` is fired simultaneously. In the example above, a firing of `move(t)` produces the removal of a term `t` from the `p1` place, and the insertion of this term into the `p2` place. It is worth noting that method `put(t)` can be fired alone, without the firing of method `move(t)`, while the firing of method `move(t)` cannot be performed without the firing of method `put(t)`. In addition, as place `p1` contains three terms, method `move(t)` can be fired at most three times; (3) transition `processing` fires spontaneously as soon as it is enabled, and fires as long as it is enabled. While transition `processing` is firing no other method can be fired, e.g. method `put(t)` cannot be fired. Transitions have a higher priority than methods. The process of ending the firing of internal transitions before firing any method is called *stabilisation process* in the CO-OPN/2 framework.

This report is a first attempt to attach real-time constraints to CO-OPN/2 specifications. In order to study how real-time constraints can be introduced into CO-OPN/2 specifications, we will work on a simplified version of CO-OPN/2 specifications. This version, called *Synchronised Petri Nets*, is such that:

- *Priority is rendered with inhibitor arcs.*
  CO-OPN/2 transitions are important, since they provide a priority mechanism of transitions wrt methods. However, the stabilisation process prohibits the guarantee of real-time constraints. Therefore, internal transitions are kept in synchronised Petri nets, but the priority involved by the stabilisation process is rendered with inhibitor arcs instead of the stabilisation process. In addition, inhibitor arcs are a more general mechanism than the stabilisation process. Indeed, the use of inhibitor arcs enables to choose the methods to be given lower priority wrt transitions, while in the case of the stabilisation process, every method is delayed until every transition ends.

- *ADT are replaced with black tokens.*
  In order to simplify the notations and the semantics, we replace ADT by black tokens. This simplification is acceptable since we want to focus on real-time constraints, and the chosen notion of time does not depend on the data type.

- *Object-based replaces object-orientation.*
  The CO-OPN/2 semantics allows the creation and destruction at run-time of Class instances. In this attempt to consider real-time constraints we will focus on synchronisation rather than creation of instances. Therefore, synchronised Petri nets specifications are made of static objects, i.e. a fixed number of Petri nets that exist since the beginning of the system.

- *Synchronisation is kept.*
  Focus is given on synchronisation and real-time constraints attached to synchronised methods. Therefore, the synchronisation mechanism of CO-OPN/2 is kept.

- *Real-time constraints are added.*
  A time interval can be attached to any method and any transition. The firing of a method or a transition is considered to be instantaneous, it takes place in a time interval that is relative to the time when the method or transition becomes enabled.

Figure 2 gives a synchronised Petri net version of the CO-OPN/2 Toy example of Figure 1.

The three algebraic terms `t1`, `t2`, and `t3` are replaced by three black tokens.

Methods `move` and `put` have no parameter attached, since we cannot discriminate among black tokens. Methods have a time interval attached: method `move` has the time interval `[5..15]` attached, method `put` has the time interval `[2..10]` attached, and transition `processing` the time interval `[1..9]`.

There is an inhibitor arc, with label `0`, between place `p2` and method `put`.

[1] transitions may also request a synchronisation with methods, however neither transitions nor methods can request a synchronisation with transitions.

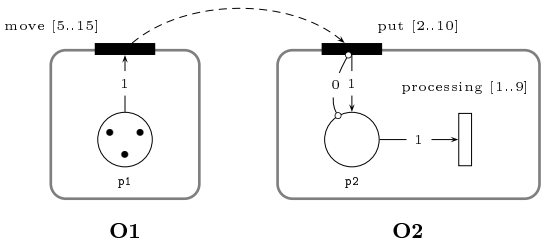


Figure 2: Toy Example with Real-time Synchronised Petri nets and Inhibitor Arcs

The intuitive semantics of this real-time synchronised Petri net is the following: (1) the time interval attached to method `move` means that method `move` must fire instantaneously in the interval given by: `5` time slots after it becomes enabled, and `15` time slots after it becomes enabled. In addition, it *must* fire at latest `15` time slots after it becomes enabled; (2) similarly for method `put` and transition `processing`: method `put` must fire not before `2` time slots after it becomes enabled, and at latest `10` time slots after it becomes enabled, while transition `processing` must fire not before `1` time slots after it becomes enabled, and at latest `9` time slots after it becomes enabled; (3) method `move` requires to be synchronised with method `put`, and their respective time intervals must be respected, this means that method `move` and method `put` must fire simultaneously in an interval corresponding to the intersection of the time interval of `move`, and that of `put`, i.e., `[2..10]` (since both are enabled at time 0); (4) the inhibitor arc associated to method `put` means that this method can be fired if the number of tokens in place `p2` is less or equal `0`. Therefore, transition `processing` has a higher priority wrt method `put`. Indeed, method `put` cannot fire, if transition `processing` has not fired before, and thus emptied place `p2`.

The goal of this report is to precise the syntax and semantics of these real-time synchronised Petri nets. The report is structured in the following manner: section 2 defines the syntax and semantics of synchronised Petri nets; section 3 defines the syntax and semantics of real-time synchronised Petri nets; and section 4 gives some applicative examples.

2 Synchronised Petri Nets

Synchronised Petri nets are a simplified version of CO-OPN/2 specifications: synchronisation of nets is maintained, but object-oriented features, ADT, and the stabilisation process are removed.

2.1 Syntax

A *synchronised Petri net* is a Petri net with two kinds of transitions: external ones, called methods, and internal ones, simply called transitions. Three kinds of arcs between places and methods/transitions are defined: input, output, and inhibitor arcs. Input, output arcs are traditional pre- post-conditions of nets. Inhibitor arcs prevent the firing of a single method or transition if the number of tokens in the place is greater than the number of tokens requested by the arc.

Definition 2.1 *Synchronised Petri Net.*
A Synchronised Petri Net is given by a 6-tuple $(P, M, T, Pre, Post, In)$ *where:* P *is a finite set of places;* M *is a finite set of methods;* T *is a finite set of (internal) transitions;* $Pre, Post : M \cup T \rightarrow (P \rightarrow \mathbb{N})$ *are total functions, they define traditional Petri nets arcs removing or inserting black tokens respectively; to every method or transition is associated a partial function that maps places to a number.* $In : M \cup T \rightarrow (P \rightarrow \mathbb{N})$ *is a total function defining inhibitor arcs*[2]*.*

A synchronised Petri nets system is a set of synchronised Petri nets with a synchronisation mapping among them.

Definition 2.2 *Synchronised Petri nets System.*
A Synchronised Petri nets System is given by $Sys = (O_1, \ldots, O_n, Sync)$*:*

[2]Synchronised Petri nets are *not* labelled nets. Indeed, methods and transitions of a synchronised Petri net have at most one behaviour. This is different from CO-OPN/2 nets, where a given method or transition is allowed to have several different behaviours.

- $O_i = (P_i, M_i, T_i, Pre_i, Post_i, In_i)$, $1 \leq i \leq n$, *a synchronised Petri net;*
- *a total function* $Sync : \cup_{i\in\{1,\dots,n\}}(M_i \cup T_i) \rightarrow Sync_{Expr}$ *that defines for each method and transition* $m \in \cup_{i\in\{1,\dots,n\}}(M_i \cup T_i)$ *a synchronisation expression* $e \in Sync_{Expr}$.

The following conditions must hold:

- *for every* $i, j \in \{1, \dots, n\}$ *then* $P_i \cap M_j = P_i \cap T_j = M_i \cap T_j = \varnothing$;
- *for every* $i, j \in \{1, \dots, n\}, i \neq j$ *then* $P_i \cap P_j = \varnothing$, *and* $M_i \cap M_j = \varnothing$, *and* $T_i \cap T_j = \varnothing$;
- *the set* $Sync_{Expr}$ *of synchronisation expressions is the least set such that:*

$$\begin{aligned} \epsilon &\in Sync_{Expr} \\ \forall M_i, i \in \{1, \dots, n\}, M_i &\subset Sync_{Expr} \\ e_1, e_2 \in Sync_{Expr} &\Rightarrow\ e_1 \ // \ e_2 \in Sync_{Expr} \\ e_1, e_2 \in Sync_{Expr} &\Rightarrow\ e_1 \ .. \ e_2 \in Sync_{Expr} \\ e_1, e_2 \in Sync_{Expr} &\Rightarrow\ e_1 \oplus e_2 \in Sync_{Expr}. \end{aligned}$$

  ϵ *stands for the empty synchronisation;*

- *the Sync function must ensure that a method does not synchronise with itself, and that the chain of synchronisations does not form cycles. Therefore, considering the relation* $<_{Sync} \subseteq \cup_{i\in\{1,\dots,n\}}M_i \times \cup_{i\in\{1,\dots,n\}}M_i$, *where :*

$$\forall m \in \cup_{i\in\{1,\dots,n\}}M_i,\ \forall m' \in Sync(m) \ \Rightarrow\ m <_{Sync} m',$$

  the relation $<^*_{Sync} \subseteq \cup_{i\in\{1,\dots,n\}}M_i \times \cup_{i\in\{1,\dots,n\}}M_i$, *defined as the transitive closure of* $<_{Sync}$, *is such that:*

$$\forall m, m' \in \cup_{i\in\{1,\dots,n\}}M_i,\ m <^*_{Sync} m' \ \Rightarrow\ m \neq m'.$$

  The membership of a method m *to a synchronisation expression* e, *denoted by* $m \in e$, *is recursively defined by:* $\forall m \in \cup_{i\in\{1,\dots,n\}}M_i, e_1, e_2 \in Sync_{Expr}$, *then:*

$$\begin{aligned} m &\in m \\ (m \in e_1 \vee m \in e_2) &\Rightarrow m \in e_1 \ // \ e_2 \\ (m \in e_1 \vee m \in e_2) &\Rightarrow m \in e_1 \ .. \ e_2 \\ (m \in e_1 \vee m \in e_2) &\Rightarrow m \in e_1 \oplus e_2. \end{aligned}$$

Remark 2.3 *A method or a transition of a net can request the synchronisation with a method of the same net or another net (provided the same method does not appear two times or more in the chain of synchronisations). In addition, it is not possible to request synchronisation with a transition.*

Notation 2.4 *Syntactical Notation.*
Let $Sys = (O_1, \dots, O_n, Sync)$ *be a synchronised Petri nets system, with* $O_i = (P_i, M_i, T_i, Pre_i, Post_i, In_i)$, *we denote:*

$$\begin{aligned} &P = \cup_{i\in\{1,\dots,n\}}P_i \\ &M = \cup_{i\in\{1,\dots,n\}}M_i \\ &T = \cup_{i\in\{1,\dots,n\}}T_i \\ &Pre : M \cup T \rightarrow (P \rightarrow \mathbb{N}) \\ &\qquad m \in M_i \cup T_i \Rightarrow Pre(m)(p) = \begin{cases} Pre_i(m)(p), p \in P_i \\ \textit{undefined, otherwise} \end{cases} \\ &Post : M \cup T \rightarrow (P \rightarrow \mathbb{N}) \\ &\qquad m \in M_i \cup T_i \Rightarrow Post(m)(p) = \begin{cases} Post_i(m)(p), p \in P_i \\ \textit{undefined, otherwise} \end{cases} \\ &In : M \cup T \rightarrow (P \rightarrow \mathbb{N}) \\ &\qquad m \in M_i \cup T_i \Rightarrow In(m)(p) = \begin{cases} In_i(m)(p), p \in P_i \\ \textit{undefined, otherwise.} \end{cases} \end{aligned}$$

In the rest of this paper we will use this notation.

Notation 2.5 *Graphical Notation.*
Let $Sys = (O_1, \ldots, O_n, Sync)$ be a synchronised Petri nets system, with $O_i = (P_i, M_i, T_i, Pre_i, Post_i, In_i)$, the corresponding graphical specification is the following: each object O_i is depicted by an oval. Inside the oval, each $p \in P_i$ is represented with a circle. On the border of the oval, each $m \in M_i$ is given by a black rectangle. Inside the border, each $t \in T_i$ is drawn with a white rectangle. Each $Pre_i(m)(p) \geq 1$ is figured with an arrow from p to m (labelled by $Pre_i(m)(p)$), each $Post_i(m)(p) \geq 1$ with an arrow from m to p (labelled by $Post_i(m)(p)$). Each $In_i(m)(p)$ is depicted with a line, ended by circles, between m and p (labelled by $In_i(m)(p)$). Each $Sync(m) = m'$ is drawn as a dashed arrow from m to m'; and each $Sync(m) = e \in Sync$ as a dashed arrow from m to every method appearing in e, with the mention of $//$, $\ldots$, or $\oplus$.

Definition 2.6 *Marking, Set of Markings.*
Let $Sys = (O_1, \ldots, O_n, Sync)$ be a synchronised Petri nets system, a marking is a total mapping $mark : P \to \mathbb{N}$. For each place $p \in P$, the marking mentions the number of black tokens in p.
We denote by $Mark$ the set of all markings of Sys.

Definition 2.7 *Sum of Markings.*
Let $Sys = (O_1, \ldots, O_n, Sync)$ be a synchronised Petri nets system, and $Mark$ be the set of all markings of Sys. The sum of two markings is given by a mapping $+_{Mark} : Mark \times Mark \to Mark$ such that:

$$\forall p \in P \;\Rightarrow\; (mark_1 +_{Mark} mark_2)(p) = mark_1(p) + mark_2(p).$$

In the rest of this paper we simply note $+$ instead of $+_{Mark}$.

Definition 2.8 *Marked Synchronised Petri Nets System.*
A marked Synchronised Petri nets system is a pair $(Sys, mark)$ where Sys is a synchronised Petri nets system, and $mark \in Mark$ is a marking for Sys. Marking $mark$ is said to be the initial marking of Sys.

Example 2.9 *Figure 2 (without the time intervals) is the graphical notation of the marked synchronised Petri nets system $(Sys, mark)$ given by:*

$$\begin{aligned}
Sys &= (O_1, O_2, Sync)\\
O_1 &= (P_1, M_1, T_1, Pre_1, Post_1, In_1),\\
O_2 &= (P_2, M_2, T_2, Pre_2, Post_2, In_2)\\
P_1 &= \{p1\}, M_1 = \{\texttt{move}\}, T_1 = \varnothing\\
P_2 &= \{p2\}, M_2 = \{\texttt{put}\}, T_2 = \{\texttt{processing}\}\\
Pre_1(\texttt{move})(\texttt{p1}) &= 1\\
Pre_2(\texttt{processing})(\texttt{p2}) &= 1\\
Post_2(\texttt{put})(\texttt{p2}) &= 1\\
In_2(\texttt{put})(\texttt{p2}) &= 0\\
Sync(\texttt{move}) &= \texttt{put}\\
Sync(\texttt{put}) &= \epsilon\\
Sync(\texttt{processing}) &= \epsilon\\
mark(\texttt{p1}) &= 3\\
mark(\texttt{p2}) &= 0.
\end{aligned}$$

2.2 Semantics

The semantics of a synchronised Petri nets system is given by a transition system. In order to build the semantics of a synchronised Petri nets system, we build first a *basic transition system* containing triples related to single transitions. We build then an *expanded transition system* containing triples related to synchronisation, parallelism, and sequence among transitions and methods. Both the basic and the expanded transition systems are built using *Structured Operational Semantics* (SOS)) *rules*. The final semantics is then given by a subset of the expanded transition system, where we retain only the triples containing *observable events*.

This section defines first observable events and events, second transition systems for synchronised Petri nets system, third the set of rules that enable to construct the basic transition system, then the set of rules for the expanded transition system. Finally, we define the semantics of a synchronised Petri nets system.

An observable event is one of the following: the firing of a method, the firing of a transition, or the parallel (//) or sequence (..) firing of two observable events, or the alternative ($\oplus$) between two observable events.

Definition 2.10 *Observable Events.*
Let Sys be a synchronised Petri nets system. The set of observable events of Sys, denoted by Obs_{Sys}*, is the least set such that:*

$$\begin{aligned} M \cup T &\subset Obs_{Sys} \\ e_1, e_2 \in Obs_{Sys} &\Rightarrow e_1 \;//\; e_2 \in Obs_{Sys} \\ e_1, e_2 \in Obs_{Sys} &\Rightarrow e_1 \;..\; e_2 \in Obs_{Sys} \\ e_1, e_2 \in Obs_{Sys} &\Rightarrow e_1 \oplus e_2 \in Obs_{Sys}. \end{aligned}$$

Definition 2.11 *Events.*
Let Sys be a synchronised Petri nets system. The set of events of Sys, denoted by Event, is the least set such that:

$$\begin{aligned} &e \in Obs_{Sys} \Rightarrow e \in Event \\ &e = m \,\text{with}\, e',\ m \in M \cup T, \text{and } e' \in Sync_{Expr} \Rightarrow e \in Event. \end{aligned}$$

An event is any observable event, but also an event of the form "m with e", where the synchronisation is explicitly required.

Definition 2.12 *Transition System.*
A transition system trs for a synchronised Petri nets system Sys is such that:

$$trs \subseteq Mark \times Event \times Mark.$$

A transition system is a set of triples consisting of two markings and an event.

Basic Transition System

Definition 2.13 gives the rules for building the basic transition system.

Definition 2.13 *Rules, Basic Transition System.*
Let Sys be a synchronised Petri nets system, the set of rules for constructing the basic transition system is given by the rules below. In these rules: $m \in M \cup T$, $e \in Sync_{Expr}$, $mark_1 \in Mark$.

The basic transition system, denoted by trs_{basic}*, is the least fixed point resulting from the application of the inference rules R1 and R2.*

$$\text{R1}\frac{\begin{array}{c} Sync(m) = \epsilon \\ In(m) \geq mark_1 \\ mark_1 \geq Pre(m) \end{array}}{(mark_1, m, mark_1 - Pre(m) + Post(m))}$$

$$\text{R2}\frac{\begin{array}{c} Sync(m) = e \\ In(m) \geq mark_1 \\ mark_1 \geq Pre(m) \end{array}}{(mark_1, m \text{ with } e, mark_1 - Pre(m) + Post(m))}$$

Remark 2.14 *In the above rules* $In(m) \geq mark$ *holds if* $In(m)(p) \geq mark(p)$ *for every* p *where* $In(m)$ *is defined.*

There are two kinds of rules for constructing the basic transition system:

- R1
  Rule R1 covers the case of the firing of a single transition or method that does not require any synchronisation. Provided the marking $mark_1$ is greater than the pre-condition Pre, and the conditions of the inhibitor arcs hold, the resulting marking after the firing, is simply given by the removal of tokens requested by the pre-condition, and the insertion of tokens requested by the post-condition.

- R2
  Rule R2 enables to build additional triples, whose event part is of the form "m with e". Rule R2 covers the case of the firing of a single transition or method that explicitly requires to be synchronised with some other methods, given by synchronisation expression e. The resulting triple is the same as Rule R1, except the event part. These triples will not be part of the final semantics, but they are useful for building the intermediate semantics.

Expanded Transition System

The expanded transition system is obtained from the basic transition system by adding triples related to synchronisation, simultaneity, sequence and alternative.

Definition 2.15 *Rules, Expanded Transition System.*
Let Sys be a synchronised Petri nets system, and trs_{basic} its basic transition system, the set of rules for constructing the expanded transition system is given by the rules below. In these rules: $m, m_i \in M \cup T$, $e \in Sync_{Expr}$, $e_1, e_2 \in Obs_{Sys}$, $mark_1, mark'_1, mark_2, mark'_2 \in Mark$.

The expanded transition system, denoted by trs_{expand}, is the least fixed point resulting from the application of the inference rules Sync, Sim, Alt.1, Alt.2, and Seq to trs_{basic}.

$$\text{Sync}\frac{\begin{array}{ll} In(m) \geq mark_1 + mark'_1 + Post^*(e) & In^*(e) \geq mark_1 + mark'_1 + Post(m) \\ (mark_1, m \text{ with } e, mark_2) & (mark'_1, e, mark'_2) \end{array}}{(mark_1 + mark'_1, m, mark_2 + mark'_2)}$$

$$\text{Sim}\frac{\begin{array}{ll} In^*(e_1) \geq mark_1 + mark'_1 + Post^*(e_2) & In^*(e_2) \geq mark_1 + mark'_1 + Post^*(e_1) \\ (mark_1, e_1, mark_2) & (mark'_1, e_2, mark'_2) \end{array}}{(mark_1 + mark'_1, e_1 \ // \ e_2, mark_2 + mark'_2)}$$

$$\text{Alt.1}\frac{(mark_1, e_1, mark_2)}{(mark_1, e_1 \oplus e_2, mark_2)} \quad \text{Alt.2}\frac{(mark_1, e_2, mark_2)}{(mark_1, e_1 \oplus e_2, mark_2)}$$

$$\text{Seq}\frac{(mark_1, e_1, mark'_1) \quad (mark'_1, e_2, mark_2)}{(mark_1, e_1 \ .. \ e_2, mark_2)}$$

Remark 2.16 *In the above rules, the following conventions are used:*

- *$In^*(e)$ is recursively defined in the following way:*

$$e = m, Sync(m) = \epsilon \ \Rightarrow \ In^*(e) = In(m)$$

$$e = m, Sync(m) = e' \ \Rightarrow \ In^*(m)(p) = \begin{cases} \min\{In(m)(p), In^*(e')(p)\}, \text{ if both are defined} \\ In(m)(p), \text{ if } In^*(e')(p) \text{ is not defined} \\ In^*(e')(p), \text{ if } In(m)(p) \text{ is not defined} \\ \text{undefined otherwise} \end{cases}$$

$$e = e_1 \ // \ e_2 \ \Rightarrow \ In^*(e)(p) = \begin{cases} \min\{In^*(e_1)(p), In^*(e_2)(p)\}, \text{ if both are defined} \\ In^*(e_1)(p), \text{ if } In^*(e_2)(p) \text{ is not defined} \\ In^*(e_2)(p), \text{ if } In^*(e_1)(p) \text{ is not defined} \\ \text{undefined otherwise} \end{cases}$$

$$e = e_1 \ .. \ e_2 \ \Rightarrow \ In^*(e) = In^*(e_1).$$

- *$Post^*(e)$ is recursively defined in the following way:*

$$\begin{aligned} e = m, Sync(m) = \epsilon &\Rightarrow Post^*(e) = Post(m) \\ e = m, Sync(m) = e' &\Rightarrow Post^*(e) = Post(m) + Post^*(e') \\ e = e_1 \;//\; e_2 &\Rightarrow Post^*(e) = Post^*(e_1) + Post^*(e_2) \\ e = e_1 \;..\; e_2 &\Rightarrow Post^*(e) = Post^*(e_1). \end{aligned}$$

The cases $In^*(e_1 \oplus e_2)$ and $Post^*(e_1 \oplus e_2)$ are not considered, because we assume that events can always be written in a "normal" form: $e = e_1 \oplus e_2 \oplus e_3 \oplus \ldots$, where e_i do not contain the $\oplus$ operator. Therefore, rules Sync, Sim, and Seq are applied on events that do not contain the $\oplus$ operator, and rule Alt can be applied independently from Sync, Sim, and Seq.

The fact that conditions on a sequence $e_1 \;..\; e_2$ are actually conditions on e_1 is motivated by the fact that we will add timing constraints to the events that will imply a further constraint on the time of occurrence of synchronised and simultaneous events. In the case of a synchronisation: m with $e_1 \;..\; e_2$, or in the case of a simultaneous event: $(e_1 \;..\; e_2) \;//\; e_3$, conditions on inhibitor arcs hold if and only if m occurs simultaneously with e_1, or e_1 occurs simultaneously with e_3 respectively.

The rules are such that:

- Sync.
  Rule Sync handles the case of the synchronisation. From two triples, one with a requested synchronisation, and one with the corresponding synchronisation, it produces a triple where the synchronisation is abstracted. An additional condition is necessary: the inhibitor arcs of m have to be greater than the sum of markings and the post condition of e, and conversely for the inhibitor arc of e.

- Sim.
  Rule Sim handles the case of simultaneity of observable events. From two triples: one for e_1 and one for e_2 it builds the triple for event $e_1 \;//\; e_2$. Additional conditions are necessary: inhibitor arcs for e_1 have to be greater than the sum of markings and post condition of e_2, and conversely for e_2.

- Alt.1, Alt.2
  Rule Alt.1 corresponds to the case where e_1 fires. Rule Alt.2 corresponds to the case where e_2 fires.

- Seq.
  Rule Seq defines triples for sequential events: from two triples whose final and initial marking correspond, it is possible to obtain the triple for their sequence.

Semantics

Once we have built the expanded transition system according to the above rules, we obtain the semantics by retaining only the triples such that:

- only observable events appear in the triple, i.e., no "m with e" appears in the triple;
- the triples contain markings reachable from the initial marking.

Definition 2.17 *Semantics of a marked synchronised Petri nets system.*
Let $(Sys, mark)$ be a marked synchronised Petri nets system, trs_{expand} be the expanded transition system obtained with the rules given in Definition 2.15. The semantics of $(Sys, mark)$, denoted by Sem, is given by the least transition system such that:

$$\begin{aligned} (mark, e, mark') \in trs_{expand} \wedge e \in Obs_{Sys} &\Rightarrow (mark, e, mark') \in Sem \\ (mark'_1, e, mark_2) \in trs_{expand} \wedge e \in Obs_{Sys} \wedge & \\ \exists mark_1, e' \; s.t. \; (mark_1, e', mark'_1) \in Sem &\Rightarrow (mark'_1, e, mark_2) \in Sem. \end{aligned}$$

Example 2.18 *Figure 3 depicts (a subset of) the tree of reachable markings of the example of Figure 2. The initial marking is made of 3 tokens in place* `p1`*, and by the empty place* `p2`*. Therefore, the root of the tree of reachable markings is given by this initial marking, denoted by $\{3, 0\}$. Given*

this marking, three observable events can occur: (1) `move`*, which leads to marking* $\{2,1\}$ *(since* `move` *requires a synchronisation with* `put`*, both fire simultaneously); (2)* `put`*, which leads to marking* $\{3,1\}$ *(*`put` *occurs alone); or (3) the alternative* `move` $\oplus$ `put`*, which produces either marking* $\{2,1\}$ *(*`move` *actually occurs) or marking* $\{3,1\}$ *(*`put` *actually occurs).*

Because of the synchronisation between `move` *and* `put`*, each time event* `move` *fires, event* `put` *fires too. In addition, each time event* `put` *occurs, the next event to occur is necessarily* `processing`*. Indeed, the inhibitor arc (with weight 0) between place* `p2` *and method* `put` *means that* `processing` *has to empty place* `p2` *before* `put` *can newly fire. Therefore, a* `move` *or a* `put` *can be followed neither by a* `put` *nor by a* `move` *(since* `move` *requires the firing of* `put`*). A* `move` *or a* `put` *can be followed only by the firing of* `processing`*.*

Rule Sim prohibits `put` // `put` *to occur, since the condition regarding the inhibitor arc would be violated. Since* `put` *cannot occur two times simultaneously, observable events* `move` // `put`*, and* `move` // `move` *cannot occur. Rule Sim would be violated in this case too.*

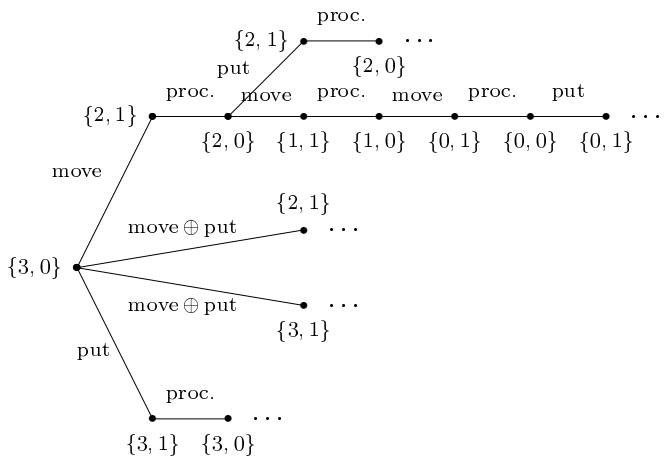


Figure 3: Tree of reachable markings

2.3 Examples

Jensen [5] defines inhibitor arcs in a different way: $In(m) \geq mark + Post(m)$. We chose instead: $In(m) \geq mark$ for single methods (or transitions) m (see Rules R1, R2), and $In(m) \geq mark_1 + mark'_1 + Post^*(e)$, when m requires a synchronisation with e (see Rules Sync, Sim).

For single methods, the choice of the condition $In(m) \geq mark$ is motivated by the fact that we want the following meaning for the weight of the inhibitor arc: a weight of 0 means that the transition (or method) attached to the inhibitor arc can fire alone provided that there are no tokens in the considered place. A weight $k > 0$ means m is able to fire alone provided that there are less or equal k tokens in the place.

Figure 4 explains the choices regarding the inhibitor arcs:

- Case a: If we remove the conditions regarding the inhibitor arcs in rule Sim ($In^*(e1) \geq mark_1 + mark'_1 + Post^*(e2)$ and $In^*(e2) \geq mark_1 + mark'_1 + Post^*(e1)$), then only local conditions would apply (this would correspond to split the resources present in places before evaluating inhibitor arcs). In this case, we could fire `m1` // `m2` but not `m1` $\oplus$ `m2` or `m1` `..` `m2`. Indeed, on one hand rule R1 can be applied to `m1` with a marking of two tokens, and to `m2` with a marking of two tokens. Then rule Sim (without conditions) could be applied to `m1` // `m2`. On the other hand, rule R1 cannot be applied to `m1` with a marking of four tokens, and consequently both `m1` $\oplus$ `m2` and `m1` `..` `m2` are not allowed.

  Allowing parallel firing, and prohibiting sequential firing is contradictory when time reduces to zero, since the simultaneity corresponds to the case when time becomes zero. Indeed, even for small times `m1` `..` `m2` is not allowed, while `m1` // `m2` would be allowed.

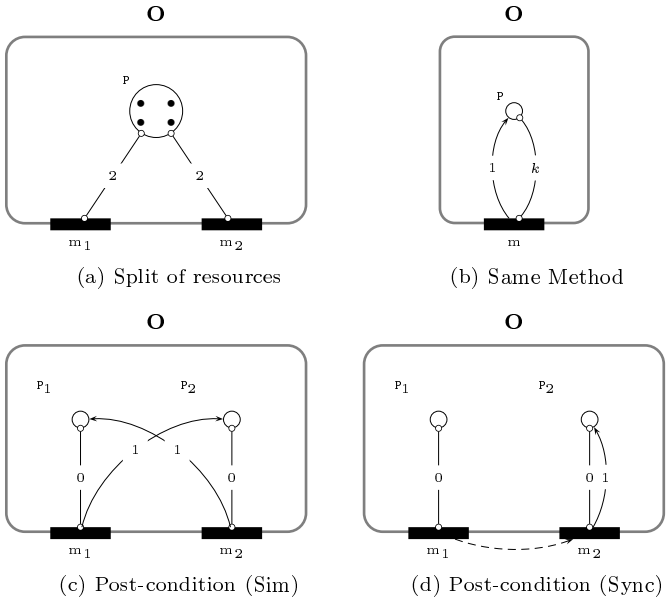
Since we will add, in a further step, timing constraints to every transition and method, we have chosen to use condition $In(e_1) \geq mark_1 + mark'_1 + Post^*(e_2)$ in rules Sync and Sim. Indeed, in the case of time, we want to avoid the fact that two events can occur in parallel (i.e., exactly at the same time t), but cannot occur in sequence, even for very short delays between the occurrences.

- Case b: In the case of a single method m, whenever the weight of the inhibitor arc is $k = 0$, since $Post^*(m)$ inserts one token in the corresponding place, then method m, cannot fire twice or more simultaneously (condition regarding the inhibitor arc would be violated in rule Sim). Whenever the weight of the inhibitor arc is $k > 0$, method m can fire n times simultaneously if $(n-1) * Post^*(m) \leq k$. In the figure below, if $k = 1$, then $n = 2$, since $m \,//\, m$ can fire, but not $m \,//\, m \,//\, m$; if $k = 2$ then $n = 3$, m can fire 3 times simultaneously, but not 4.

  Similarly to Case a, method m can fire n times simultaneously only if it is able to fire n times in sequence.

- Case c: If we had $In^*(e1) \geq mark_1 + mark'_1$ instead of $In^*(e1) \geq mark_1 + mark'_1 + Post^*(e_2)$ in rule Sim, we could fire `m1 // m2` but not `m1 .. m2`. For the same reason of Case a, we want to avoid such discontinuity when the time reduces to zero.

- Case d: If we had $In^*(e) \geq mark_1 + mark'_1 + Post^*(m)$ instead of $In^*(e) \geq mark_1 + mark'_1 + Post(m)$ in rule Sync, we could fire `m2` alone, but not `m1` (which requires a synchronisation with `m2`). Requiring $Post^*(m1)$ would require also $Post^*(m2)$ in the computation of the inhibitor arc. This would be contradictory with rule R1, where the firing of `m2` alone does not require $Post(m2)$. The chosen rule for the inhibitor arc is such that the firing of the synchronous event fails only if the method requesting the synchronisation is directly involved in the post-set of the required method.



(a) Split of resources (b) Same Method

(c) Post-condition (Sim) (d) Post-condition (Sync)

Figure 4: Inhibitor Arcs

3 Real-Time Synchronised Petri Nets

Real-time synchronised Petri nets are synchronised Petri nets with time interval attached to methods and transitions.

3.1 Syntax

A *real-time synchronised Petri net* is simply a pair given by a synchronised Petri net and a function that associates to every method and transition of the net a time interval. The method or transition has to fire in the given time interval, relatively to its time of enabling. If a method or transition is not constrained by time, then an infinite time interval is attached.

Definition 3.1 *Real-Time Synchronised Petri Net.*
A Real-Time Synchronised Petri Net is given by a pair $(O, Time)$ *where* $O = (P, M, T, Pre, Post, In)$ *is a synchronised Petri net, and* $Time$ *is a total function that associates a time interval to every method and transition of* O*:* $Time : M \cup T \to \mathbb{R}^+ \times (\mathbb{R}^+ \cup \infty)$*. The following condition must hold:*

$$Time(m) = (t_1, t_2) \;\Rightarrow\; ((t_2 \geq t_1) \vee (Time(m) = (t, \infty))).$$

Definition 3.2 *Real-Time Synchronised Petri nets System.*
A Real-Time Synchronised Petri nets System is given by $Sys = (O_1, \ldots, O_n, Sync)$*:*

- $O_i = ((P_i, M_i, T_i, Pre_i, Post_i, In_i), Time_i)$*,* $1 \leq i \leq n$*, a real-time synchronised Petri net;*
- *a total function* $Sync : \cup_{i \in \{1,\ldots,n\}}(M_i \cup T_i) \to Sync_{Expr}$ *that defines for each method and transition* $m \in \cup_{i \in \{1,\ldots,n\}}(M_i \cup T_i)$ *a synchronisation expression*[3] $e \in Sync_{Expr}$*.*

Notation 3.3 *Syntactical Notation.*
Let $Sys = (O_1, \ldots, O_n, Sync)$ *be a real-time synchronised Petri nets system, with* $O_i = ((P_i, M_i, T_i, Pre_i, Post_i, In_i), Time_i)$*, we use the same notation as 2.5 for* $P, M, T, Pre, Post, In$*. In addition, we denote*

$$Time : M \cup T \to \mathbb{R}^+ \times (\mathbb{R}^+ \cup \infty)$$
$$m \in M_i \cup T_i \Rightarrow Time(m) = Time_i(m).$$

In the rest of this paper we will use this notation.

The marking of a real-time synchronised Petri nets system is a mapping that associates to every place a multiset of real numbers. Every token of the net is stamped with its arrival time.

Definition 3.4 *Marking, Set of Markings.*
Given a real-time synchronised Petri nets system $Sys = (O_1, \ldots, O_n, Sync)$*, a marking is a total mapping:*

$$mark : P \to [\mathbb{R}^+].$$

We denote by $Mark$ *the set of all markings of* Sys*.*

A multiset of $\mathbb{R}^+$ is given by a function $f \in [\mathbb{R}^+]$ such that $f : \mathbb{R}^+ \to \mathbb{N}$, which evaluates to zero, except on a finite number of cases (i.e., the set made of all t such that $f(t) \neq 0$ is a finite set, thus the number of tokens in a place is finite). Here, $f(t) = j$ means that j tokens arrived at time t. We denote by $\varnothing$ the empty multiset ($\varnothing(t) = 0$ for every t), and for instance $\{t_1, t_1, t_2, t_2, t_2\}$ a non-empty multiset containing two tokens arrived at time t_1 and three tokens arrived at time t_2.

Definition 3.5 *Sum of Markings.*
Let $Sys = (O_1, \ldots, O_n, Sync)$ *be a real-time synchronised Petri nets system , and* $Mark$ *be the set of all markings of* Sys*. The sum of two markings is given by a mapping* $+_{Mark} : Mark \times Mark \to Mark$ *such that:*

$$(mark_1 +_{Mark} mark_2)(p) = mark_1(p) +_{[\mathbb{R}^+]} mark_2(p).$$

For every $t \in \mathbb{R}^+$*,* $(mark_1(p) +_{[\mathbb{R}^+]} mark_2(p))(t) = mark_1(p)(t) + mark_2(p)(t)$*.*
In the rest of this paper we simply note $+$ *instead of* $+_{Mark}$*, and* $+_{[\mathbb{R}^+]}$*.*

[3] $Sync_{Expr}$ is given by Definition 2.2

Definition 3.6 *Number of Elements in a Marking.*
Let mark be a marking. The number of elements in marking mark, denoted by #mark, is a total mapping:

$$\#mark : P \to \mathbb{N}, s.t. \#mark(p) = \begin{cases} \sum_{t \in K_p} mark(p)(t) \\ 0, if\, mark(p)(t) = 0, \forall t \in \mathbb{R}^+ \end{cases}$$

where $K_p = \{t \in \mathbb{R}^+ \mid mark(p)(t) > 0\}$.

The function $\#mark$ returns for every place p the number of tokens present in the place. The sum is finite since the multiset $mark(p)$ has only a finite number of elements (hence K_p is a finite set).

Definition 3.7 *Initial Marking.*
Given a real-time synchronised Petri nets system $Sys = (O_1, \ldots, O_n, Sync)$*, an initial marking is a marking,* $mark : P \to [\mathbb{R}^+]$*, such that for every* $p \in P$*:*

$$mark(p)(0) = k_p$$
$$mark(p)(t) = 0, \forall t > 0.$$

The initial marking is such that a place p contains k_p tokens arrived at time 0. The places do not contain tokens arrived after time 0.

Definition 3.8 *Marked Real-Time Synchronised Petri Nets System.*
A marked Synchronised Petri nets system is a pair $(Sys, mark)$ *where Sys is a real-time synchronised Petri nets system, and mark is an initial marking for Sys.*

Example 3.9 *Figure 2 (with the time intervals) is the graphical notation of the marked real-time synchronised Petri nets system* $(Sys, mark)$ *given by:*

$$\begin{aligned}
Sys &= (O_1, O_2, Sync) \\
O_1 &= (P_1, M_1, T_1, Pre_1, Post_1, In_1, Time_1), \\
O_2 &= (P_2, M_2, T_2, Pre_2, Post_2, In_2, Time_2) \\
P_1 &= \{p1\}, M_1 = \{\texttt{move}\}, T_1 = \varnothing \\
P_2 &= \{p2\}, M_2 = \{\texttt{put}\}, T_2 = \{\texttt{processing}\} \\
Pre_1(\texttt{move})(\texttt{p1}) &= 1 \\
Pre_2(\texttt{processing})(\texttt{p2}) &= 1 \\
Post_2(\texttt{put})(\texttt{p2}) &= 1 \\
In_2(\texttt{put})(\texttt{p2}) &= 0 \\
Sync(\texttt{move}) &= \texttt{put} \\
Sync(\texttt{put}) &= \epsilon \\
Sync(\texttt{processing}) &= \epsilon \\
Time_1(\texttt{move}) &= (5, 15) \\
Time_2(\texttt{put}) &= (2, 10) \\
Time_2(\texttt{processing}) &= (1, 9) \\
mark(\texttt{p1}) &= \{0, 0, 0\} \\
mark(\texttt{p2}) &= \varnothing.
\end{aligned}$$

3.2 Semantics

In order to build the semantics of synchronised Petri nets systems, we build first, using an initial set of rules, a *weak transition system* that contains transitions belonging to the weak time semantics (an enabled transition may not fire even if the time of occurrence elapses). Second, on the weak transition system, we apply a condition that enables to retain only those transitions that belong to the strong time semantics (an enabled transition *must* fire when the time of occurrence elapses). We obtain what we call the *strong transition system*. Third, on the strong transition system, we apply another set of rules (taking into account synchronisations) that enables us to obtain an

expanded transition system. Finally, we retain only those transitions necessary for the (observable) *strong time semantics.*

Transition systems for real-time synchronised Petri nets are made of 4-tuples (instead of triples): the time of occurrence of the transition is added.

Definition 3.10 *Transition System.*
A transition system, trs, for a real-time synchronised Petri nets system Sys is such that:

$$trs \subseteq Mark \times Event \times Mark \times \mathbb{R}^+.$$

Event is given by Definition 2.11, and Mark by Definition 3.4.

Weak Transition System

The rules for constructing the weak transition system of a real-time synchronised Petri net are given by Definition 3.11 below:

Definition 3.11 *Rules, Weak Transition System.*
Let Sys be a real-time synchronised Petri nets system, the set of rules for constructing the weak transition system is given by the rules below. In these rules: $m, m_i \in M \cup T$; $e \in Sync_{Expr}$, and $mark_1, mark \in Mark$ are markings.

The weak transition system, denoted by trs_{weak}, is the least fixed point obtained by the application of the inference rules R1.a, R1.b, R2.a, R2.b below.

$$\text{R1.a}\frac{\begin{array}{l} Sync(m) = \epsilon \\ In(m) \geq \#mark_1 \\ Time(m) = (t_1, t_2) \\ t \in [t_1 + x, t_2 + x] \wedge x = \max(mark) \\ \#mark = Pre(m) \\ mark_1 \geq mark \end{array}}{(mark_1, m, mark_1 - mark + Post(m)_t, t)}$$

$$\text{R1.b}\frac{\begin{array}{l} Sync(m) = \epsilon \\ In(m) \geq \#mark_1 \\ Time(m) = (t_1, \infty) \\ t \geq t_1 + x \wedge x = \max(mark) \\ \#mark = Pre(m) \\ mark_1 \geq mark \end{array}}{(mark_1, m, mark_1 - mark + Post(m)_t, t)}$$

$$\text{R2.a}\frac{\begin{array}{l} Sync(m) = e \\ In(m) \geq \#mark_1 \\ Time(m) = (t_1, t_2) \\ t \in [t_1 + x, t_2 + x] \wedge x = \max(mark) \\ \#mark = Pre(m) \\ mark_1 \geq mark \end{array}}{(mark_1, m \text{ with } e, mark_1 - mark + Post(m)_t, t)}$$

$$\text{R2.b}\frac{\begin{array}{l} Sync(m) = e \\ In(m) \geq \#mark_1 \\ Time(m) = (t_1, \infty) \\ t \geq t_1 + x \wedge x = \max(mark) \\ \#mark = Pre(m) \\ mark_1 \geq mark \end{array}}{(mark_1, m \text{ with } e, mark_1 - mark + Post(m)_t, t)}$$

Remark 3.12 *In the above rules, the following conventions are used:*

- $In(m) \geq \#mark$ *holds if* $In(m)(p) \geq \#mark(p)$ *for every* p *where* $In(m)$ *is defined (*$\#mark(p)$ *stands for the number of elements in* $mark(p)$*).*
- $\#mark = Pre(m)$ *holds if* $\#mark(p) = Pre(m)(p)$ *for every* $p \in P$*;*
- $Post(m)_t : P \rightarrow [\mathbb{R}^+]$ *is a marking such that all tokens are stamped at time* t*:*

$$Post(m)_t(p)(t) = Post(m)(p)$$
$$Post(m)_t(p)(t') = 0, \forall t' \neq t.$$

- $x = \max(mark)$ *stands for:*

$$x = \max(\{0\}, \cup_{p \in P} K_p),$$

x is the time when the last token that will be removed (i.e., in $mark$) arrived in the net (remember that $K_p = \{t \in \mathbb{R}^+ \mid mark(p)(t) > 0\}$*).*

Remark 3.13 *Rules 3.11 provide the weak time semantics since nothing forces enabled methods (or transitions) to fire within the given time interval.*

The rules are such that:

- R1.a and R1.b
  Rules R1.a and R1.b cover the case of the firing of a single transition or method that does not require any synchronisation. R1.a is for a finite time interval, while R1.b is for an infinite time interval. The 4-tuple is part of the weak transition system if the time of occurrence is in the absolute time interval. The absolute time interval is computed from the relative one (given by $Time(m)$) and by the greatest time of arrival of the tokens that will be removed ($\max(mark)$). Marking $mark$ stands for the real-time marking that will be removed from the places when the transition fires. Of course the number of tokens that will be removed must match the pre-condition, i.e. $\#mark = Pre(m)$.
- R2.a and R2.b
  Rules R2.a and R2.b enables to build additional 4-tuples. They cover the case of the firing of a single transition or method that requires to be synchronised with some other methods, given by synchronisation expression e. The resulting 4-tuple is the same as Rule R1.a and R1.b, except the event part.

Strong Transition System

The strong transition system is obtained as a subset of the weak transition system, by applying a condition that removes the tuples containing a transition that fires at a time such that it prevents the firing of another transition, whose time of occurrence elapsed.

Definition 3.14 *Strong Transition System.*
Let Sys be a real-time synchronised Petri nets system, and trs_{weak} *be its weak transition system. The strong transition system, denoted by* trs_{strong}*, is the maximal subset of* trs_{weak} *such that:*

$$\forall (mark_1, e, mark_2, t) \in trs_{strong} \;\Rightarrow\; Cond(mark_1, mark_2) \text{ holds.}$$

Conditions Cond is such that:

$$\begin{aligned} Cond(mark_1, mark_2) ==& \nexists\, (m', mark', t') \in (M \cup T) \times Mark \times \mathbb{R}^+ \text{ s.t.} \\ & t' < t \;\wedge \\ & Time(m') = (t'_1, t'_2) \;\wedge \\ & t' = (\max(mark') + t'_2) \;\wedge \\ & ((mark_1, m', mark_1 - mark' + Post(m')_{t'}, t') \in trs_{weak} \;\vee \\ & (mark_1, m' \text{ with } e', mark_1 - mark' + Post(m')_{t'}, t') \in trs_{weak})). \end{aligned}$$

Remark 3.15 *Condition Cond removes from the weak transition system, every 4-tuple, of the form* $(mark_1, e, mark_2, t)$*, for which there exists another 4-tuple that should have fired before.*

Therefore, condition Cond guarantees the strong time semantics: a transition must fire at time t *if it is enabled at* t*, and if time* t *is the maximal bound of firing of the transition. However, when two transitions or more reaches their maximal bound of firing at the same time, one of them may still disable the others.*

Expanded Transition System

The expanded transition system is obtained from the strong transition system by adding tuples regarding synchronisation, simultaneity, sequence and alternative.

Definition 3.16 *Rules, Expanded Transition System.*
Let Sys be a real-time synchronised Petri nets system, the set of rules for constructing the expanded transition system is given by the rules below. In these rules: $m, m_i \in M \cup T$*;* $e \in Sync_{Expr}$*, and* $mark_1, mark_2, mark'_1, mark'_2, mark \in Mark$ *are markings.*

The expanded transition system, denoted by trs_{expand}*, is the least fixed point obtained by the application, to* trs_{strong}*, of the inference rules Sync, Sim, Alt.1, Alt.2, and Seq below.*

$$\text{Sync}\frac{\begin{array}{ll} In(m) \geq \#mark_1 + \#mark'_1 + Post^*(e) & In^*(e) \geq \#mark_1 + \#mark'_1 + Post(m) \\ (mark_1, m \text{ with } e, mark_2, t) & (mark'_1, e, mark'_2, t) \end{array}}{(mark_1 + mark'_1, m, mark_2 + mark'_2, t)}$$

$$\text{Sim}\frac{\begin{array}{ll} In^*(e_1) \geq \#mark_1 + \#mark'_1 + Post^*(e_2) & In^*(e_2) \geq \#mark_1 + \#mark'_1 + Post^*(e_1) \\ (mark_1, e_1, mark_2, t) & (mark'_1, e_2, mark'_2, t) \end{array}}{(mark_1 + mark'_1, e_1 \ // \ e_2, mark_2 + mark'_2, t)}$$

$$\text{Alt.1}\frac{(mark_1, e_1, mark_2, t)}{(mark_1, e_1 \oplus e_2, mark_2, t)} \quad \text{Alt.2}\frac{(mark_1, e_2, mark_2, t)}{(mark_1, e_1 \oplus e_2, mark_2, t)}$$

$$\text{Seq}\frac{\begin{array}{ll} & t' \geq t \\ (mark_1, e_1, mark'_1, t) & (mark'_1, e_2, mark_2, t') \end{array}}{(mark_1, e_1 \ .. \ e_2, mark_2, t)}$$

Remark 3.17 $In^*(e)$ *and* $Post^*(e)$ *are defined in the same manner as in the case without time.*

As already mentioned in the case without time, constraints on time of firing of synchronous and simultaneous events are such that: an event m with $e_1 \ .. \ e_2$ *occurs at time* t*, if* m *occurs at time* t *and* $e_1 \ .. \ e_2$ *occurs at time* t*; event* $e_1 \ .. \ e_2$ *occurs at time* t*, if* e_1 *occurs at time* t *and* e_2 *occurs after* t*. Event* $(e_1 \ .. \ e_2) \ // \ e_3$ *occurs at time* t*, if* e_1 *and* e_3 *occur at time* t*, and* e_2 *occurs after.*

The CO-OPN/2 original definition of such synchronised and simultaneous event allows more non determinism. In the case of $(e_1 \ .. \ e_2) \ // \ e_3$*,* e_3 *occurs at some time between the beginning of* e_1 *and the end of* e_2*. If in a particular case, we want to allow more non determinism in the time of occurrence of transitions in synchronous and simultaneous events, we use other events. For instance, we will use* $((e_1 \ // \ e_3) \ .. \ e_2) \oplus (e_1 \ .. \ (e_2 \ // \ e_3))$ *instead of* $(e_1 \ .. \ e_2) \ // \ e_3$*.*

These rules are such that:

- Sync.
  Rule Sync handles the case of the synchronisation. From two 4-tuples, one with a requested synchronisation, and one with the corresponding synchronisation that occur *at the same time* t, it produces a 4-tuple, occurring at t, where the synchronisation is abstracted.

- Sim.
  Rule Sim handles the case of simultaneity of observable events. From two 4-tuples: one for e_1 and one for e_2 that occur *at the same time* t, it builds the 4-tuple for event $e_1 \ // \ e_2$.

- Alt.1, Alt.2
  Rule Alt.1 corresponds to the case where e_1 fires. Rule Alt.2 corresponds to the case where e_2 fires.

- Seq.
  Rule Seq defines 4-tuples for sequential events: from two 4-tuples whose final and initial marking correspond, and whose time of occurrence of the second is greater than the time of occurrence of the first one, it is possible to obtain the 4-tuple for their sequence.

Strong Time Semantics

The semantics of a real-time synchronised Petri nets system is obtained by retaining, from trs_{expand}, the 4-tuples such that:

- only observable events appear in the 4-tuple, i.e., no m with e appears in the 4-tuple;
- 4-tuples contain markings reachable from the initial marking.

Definition 3.18 *Strong Time Semantics of a marked real-time synchronised Petri nets system. Let $(Sys, mark)$ be a marked real-time synchronised Petri nets system, trs_{expand} be the expanded transition system obtained with the rules of Definition 3.16. The semantics of $(Sys, mark)$, denoted by Sem, is given by the least transition system such that:*

$$(mark, e, mark', t) \in trs_{expand} \wedge e \in Obs_{Sys} \quad \Rightarrow \ (mark, e, mark', t) \in Sem$$
$$(mark'_1, e, mark_2, t) \in trs_{expand} \wedge e \in Obs_{Sys} \wedge \exists mark_1, e', t' \ s.t.$$
$$(mark_1, e', mark'_1, t') \in Sem \ \wedge t' \leq t \quad \Rightarrow (mark'_1, e, mark_2, t) \in Sem.$$

Remark 3.19 *If we want to obtain the weak time semantics, instead of the strong time semantics, we proceed from the weak transition system 3.11, then we apply the rules of Definition 3.16, without applying the condition Cond, and then we apply Definition 3.18.*

Figure 5 shows a partial view of the tree of reachable markings of the strong time semantics of the real-time synchronised Petri nets system of Figure 2.

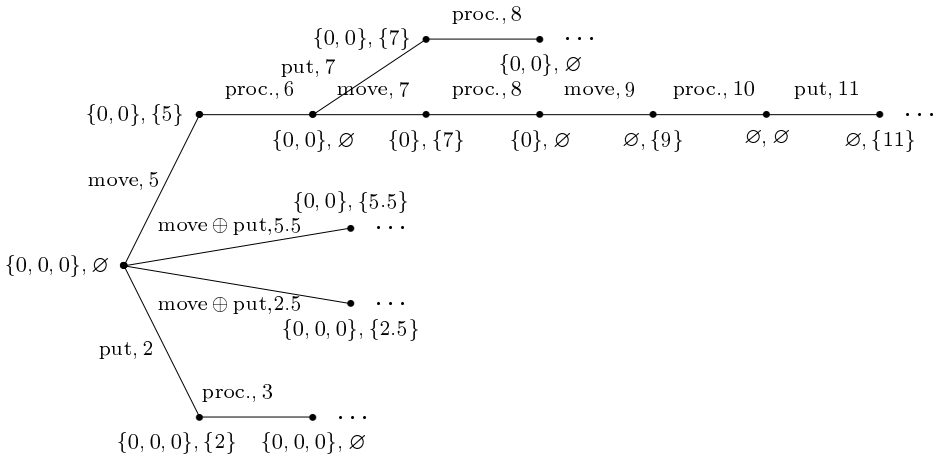


Figure 5: Tree of reachable markings

The tree is similar to that of Figure 3, except that time of occurrences of transitions and methods are denoted. The initial marking is made of three tokens arrived at time 0 in place `p1`, and the empty place `p2`, it is denoted $\{0, 0, 0\}, \varnothing$. Method `move` has to occur between time 5 and time 10 (because of `put`), and method `put` has to occur between time 2 and time 10. The figure shows the case of method `move` occurring at time 5, this leads to a new marking where a token stamped at 5 is in place `p2`.

It is worth noting that the tree depicted by Figure 5 is not complete. Indeed, from the initial marking it is also possible to fire method `move` at any time in the interval $[5, 10]$; and to fire method `put` at any time in the interval $[2, 10]$. Since method `put` has no pre-set, the relative time interval $[2, 10]$ is also the absolute time interval: R1.a applies to `put` with $x = \max(mark) = 0$,

and therefore the time t of occurrence must be such that $t \in [2, 10]$. Similarly, from any other reachable marking, method `move` and `put` can fire at any time in the interval $[5, 10]$, respectively $[2, 10]$ *after* transition `processing` has emptied place `p2`. For instance, if `processing` occurs at time 6, then method `put` can occur at any time in the interval $[6, 10]$.

3.3 Examples

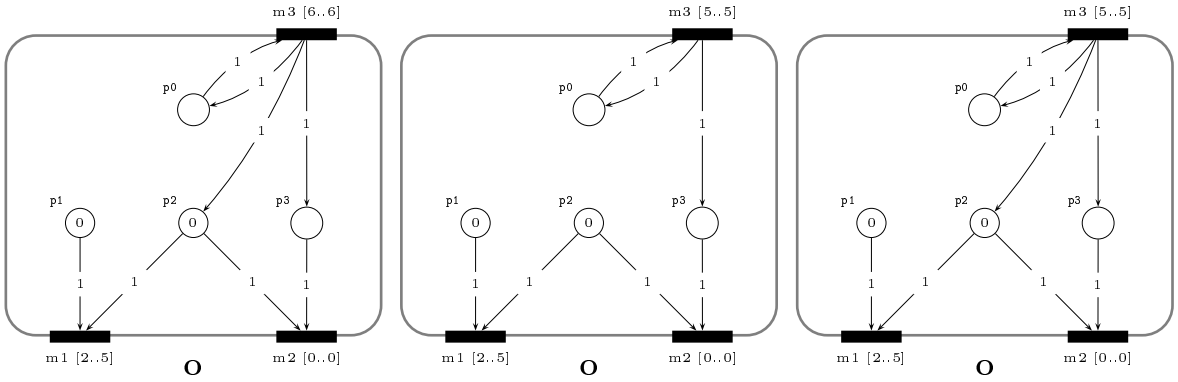
Figure 6 shows three cases related to condition *Cond* (weak time semantics wrt strong time semantics). The initial marking is such that places `p1` and `p2` have one token each, stamped with time 0:

- Case a: *`m1` must fire at 5.*
  In the weak time semantics it may occur that method `m1` does not fire at 5, and that method `m3` fires at 6, followed immediately by the firing of method `m2` at 6.

  In the strong time semantics, condition *Cond* removes from the tree of reachable markings, the firing of method `m3` at 6 if it is not preceded by the firing of method `m1` (at 5 or before).

- Case b: *`m1` or `m2` must fire at 5, the firing of one of them disables the other.*
  Method `m2` must fire immediately when a token reaches place `p3`, i.e., when method `m3` fires (at 5). In the strong time semantics, two cases occur: either `m3` and `m1` fire both at 5, and `m2` cannot fire because `m1` removes the token from `p2`; or `m3` fires at 5, `m2` fires immediately after `m3` (at 5 too), and `m1` cannot fire because `m2` removes the token from `p2`.

- Case c: *`m1` and `m2` must fire at 5.*
  Since method `m3` inserts a token into place `p2`, the disabling of method `m2` by method `m1` or vice-versa does not occur. In the strong time semantics, we have that the three methods fires at 5.



(a) Must fire (b) Disabling (c) Both Must fire

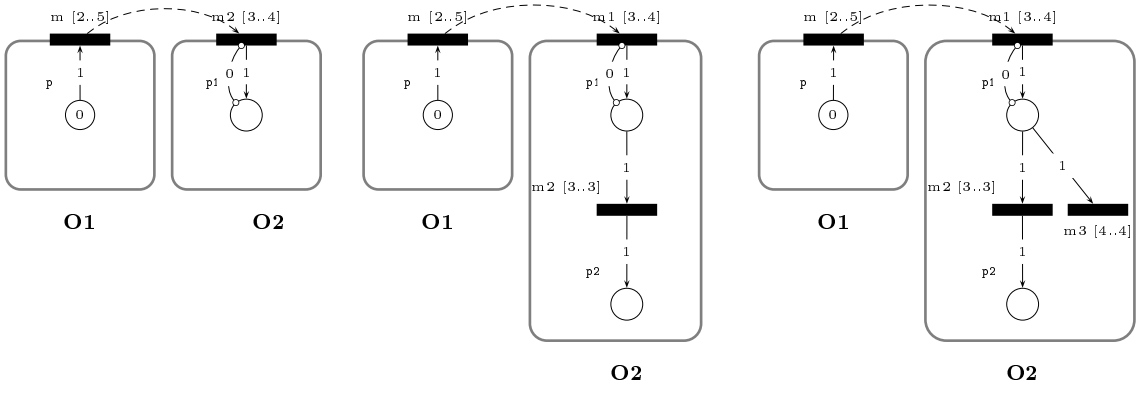
Figure 6: Simple Methods

Figure 7 shows three cases related to synchronisation and sequential firings:

- Case a: *`m` must fire in* [3..4]
  Method `m` requires the firing of `m2`. Since the time of firing of `m2` is [3..4], and that of `m` is [2..5], the intersection gives [3..4]. Since it has no pre-set, method `m2` cannot fire after time 4.

- Case b: *if `m` fires at* 3.5*, the firing of `m` produces a token stamped with time* 6.5 *in place `p2`.*
  If method `m` fires at 3.5, thus method `m2` becomes enabled at 3.5. Since it must fire in the time interval [3..3], this means that `m2` must fire at 6.5.

- Case c: *`m3` never fires.*
  Method `m3` becomes enabled at the same time as `m2`. Since `m2` must fire exactly 3 times slot after it becomes enabled, and `m3` must fire exactly 4 times slot after it becomes enabled, `m2` always fires before `m3`. Therefore, `m3` never fires.

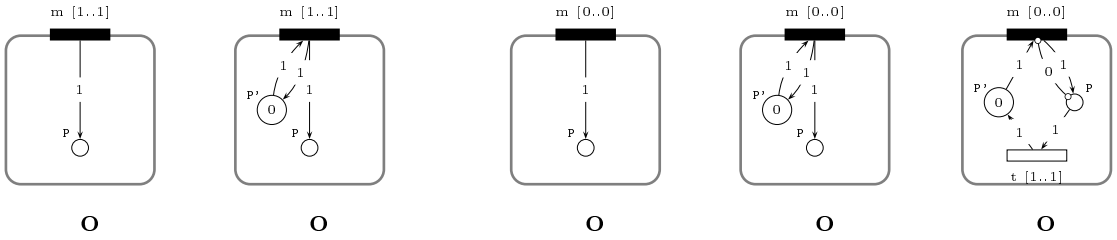


(a) Simple (b) Sequence (c) Disabling

Figure 7: Synchronisation and Sequence

Figure 8 shows two cases related to an uncountable number of firings. The semantics given above allows cases, where the number of firings of a transition or a method may be uncountable. These cases occur for transitions or methods having no pre-set, and no inhibitor arc, or when intervals [0..0] are used.

- Case a: *Method* `m` *fires an uncountable number of times at time* 1.
  Rule R1 applied to method `m` provides triples $(\varnothing, \mathtt{m}, \{1\}, 1)$, $(\{1\}, \mathtt{m}, \{1,1\}, 1)$, $(\{1,1\}, \mathtt{m}, \{1,1,1\}, 1)$, etc. The final semantics Sem, allows all these triples, since they derive all from the initial marking.
- Case b: *Method* `m` *fires only once at* 1, 2, *etc.*
  In this case, method `m` requires a pre-set in place `P'`. Rule R1 implies that `m` fires at 1, then it must wait 1 unit of time before firing, since the new token in `P'` is stamped with time 1.
- Case c: *Method* `m` *fires an uncountable number of times at time* 0.
  For the same reason as Case a, method `m` fires at 0.
- Case d: *Method* `m` *still fires an uncountable number of times at time* 0. The solution found in case b to avoid uncountable number of firings cannot be applied for intervals of the form [0..0]. Indeed, rule R1, applied to case d, provides the same triples as in case c, since `m` will fire immediately after the new token arrives in place `P'`, i.e. at time 0.
- Case e: *Method* `m` *fires at* 0, 1, *etc.*
  In this case, the transition `t` requires 1 unit of time for emptying place `P`, and the inhibitor arc prevents `m` from firing at 0 if place `P` is not empty.



(a) Uncountable (b) Countable (c) Uncountable (d) Uncountable (e) Countable

Figure 8: Uncountable Number of Firings

Figure 9 shows several small examples illustrating the chosen semantics for real-time synchronised Petri nets.

- Case a: *Method m fires in [t1..t2].*
  Method m can fire at any time between the absolute interval [t1..t2], it cannot fire after t2.
- Case b: *Method m fires in [t1..∞]*
  Method m can fire at any time between the absolute interval [t1..∞], it cannot fire before t1.
- Case c: *Method m waits at least t1, and at most t2 between two firings.*
  Because of the pre-set in place P, the time interval is re-evaluated after each new token in P. The strong time semantics implies that m must fire at the end of the time interval.
- Case d: *Method m waits at least t1 between two firings.*
  This case is similar to Case c, except that there is no upper bound for the firing.
- Case e: *Method m fires only once.*
  Method m must fire in the time interval [t1..t2]. Once it has fired, the inhibitor arc prevents any subsequent firing of m.

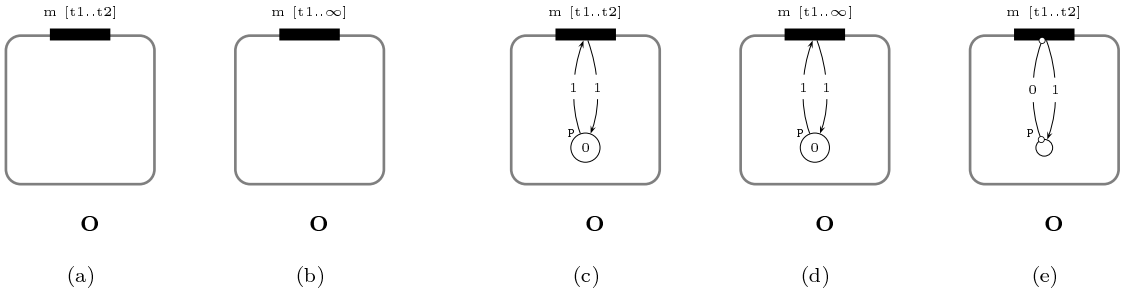


Figure 9: Illustrative Examples

3.4 Choices and Alternatives

We have already seen in Section 2 that providing a precise semantics to inhibitor arcs requires some critical choices even for untimed nets. Things become even more intricate -and somewhat controversial- when we augment nets with timing behaviour. The main point is relating the absence of a token in an inhibited place with the time interval during which a transition is enabled to fire. To illustrate, consider Figure 10.

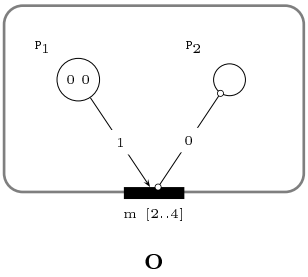


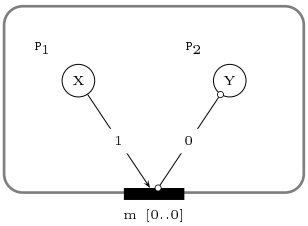
Figure 10: Inhibitor Arcs and Time (1)

A possible semantics (we call it SEM1) to be given to such a net is that transition enabling is evaluated on the basis of the "positive" conditions only, i.e., on the marking of P_1. On this basis, the method should fire at a time between 2 and 4: at that time we verify whether P_2 has tokens or not and, in the negative case, m is actually fired. Notice that, in this way, m could fire twice simultaneously at any time between 2 and 4 (provided P_2 is empty) thanks to the double enabling provided by the two tokens in P_1. On the other hand, suppose that a token abides in P_2 continuously from 1 to 5. Then, when P_2 is emptied, the timeout for m to fire is expired so that the tokens in P_1 are "dead" producing a situation somewhat similar to weak time semantics.

An alternative semantics (we call it SEM2) instead would interpret the lack of tokens in inhibited places in a more symmetric way wrt positive arcs: roughly speaking no tokens in a inhibited

place are assimilated to the presence of tokens in places with normal arcs. This requires the knowledge of the time when the place has been emptied to define precisely the time semantics. According to SEM2, and still with reference to Figure 10, if we assume that no token ever occurs in P_2, then we can have a first firing of m at a time between 2 and 4. After this firing, however, we must wait again at least 2 time units -and no more than 4- before m can fire again.

Both SEM1 and SEM2 have pros and cons. SEM1 is easier to formalise since it does not require the knowledge of "negative time stamps", i.e. recording the time when a token has been taken out from a place. There are cases, however, when definitely SEM2 better fits practical needs. For instance, in a situation such as that in Figure 11 one may typically wish to formalise that token X in P_1 waits to fire method m as soon as P_2 is emptied: with SEM1 instead, token X would be dead unless P_2 is already empty at time X. On the other hand, with SEM2 there would be no way of firing a transition more than once simultaneously if it has inhibitor arcs (at least if they have weight 0, as it will be illustrated later).



O

Figure 11: Inhibitor Arcs and Time (2)

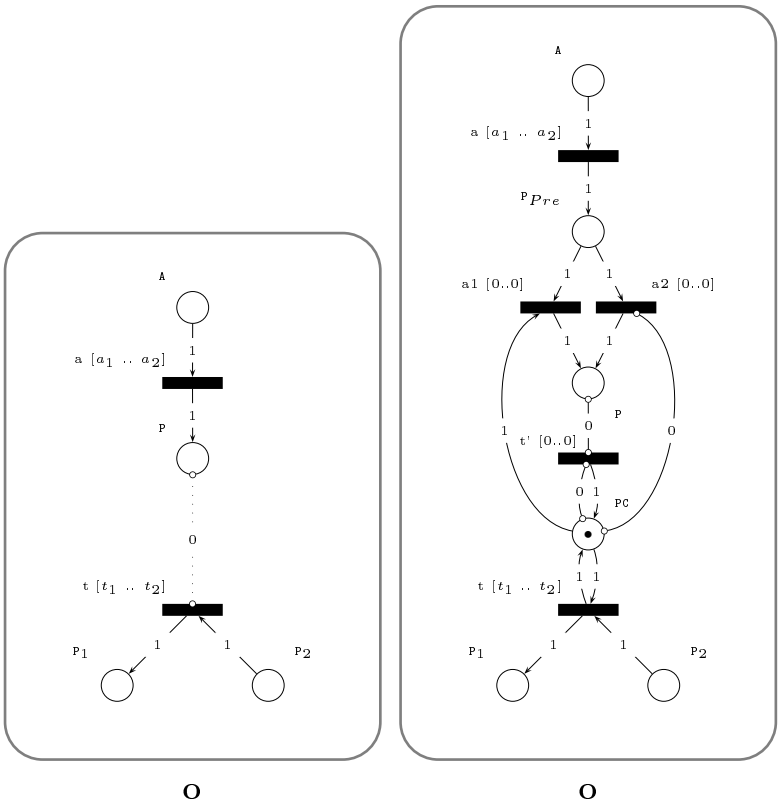
On the basis of these remarks we chose to adopt SEM1 as the basic semantics. SEM2 however, can be adopted as a *derived semantics* by introducing a different type of inhibitor arcs (with dotted line) as a short notation for a more complex net as illustrated in Figure 12 (in the case that inhibitor arcs have weight 0). The main idea underlying the construction of Figure 12 is to build a "complement place" PC attached to the original place P so that we have a token in PC whenever P is empty and conversely. Furthermore, all methods, such as a, that put tokens into P are split in such a way that, first, through a1 they empty PC and put a first token in P; for all other tokens method a2 is used instead (notice that a1 and a2 are mutually exclusive). Thus, the net of Figure 12 (b) provides SEM2 to the net of Figure 12 (a).

The symmetry between the presence of tokens in normal places and their absence in places with inhibitor arcs can be further pursued by exploiting weights attached to inhibitor arcs as well as those attached to positive arcs. In fact, intuitively, we can consider a place P with k tokens and an inhibitor arc with weight k as a place whose contents -or lack of contents- provides enough resources to its output method to let it fire just once. Instead, if P contains only k−1 tokens this enables its output method twice in the same way as two tokens enable twice a method which is connected to a place by one normal arc (with weight 1); and so on with fewer and fewer tokens. Of course the symmetry is eventually broken by the impossibility of having a negative marking. This interpretation however, seems to be general enough so that SEM1 and SEM2, together with the use of k-weighted arcs, cover all cases of practical interest. Figure 13 illustrates the natural generalisation of the construction of figure 12 to the case of weighted inhibitor arcs. In this case we have K_{max} +1 tokens in place PC.

4 Applicative Examples: Trains

In this section we illustrate the applicability of the real-time synchronised Petri nets to practical cases through a fairly classical "benchmark" for real-time system formalisms, i.e., the Generalised Railroad Crossing (GRC) system [4]. First, we informally describe the GRC system and its properties. Then, we show how it can be formalised through real-time synchronised Petri nets. Finally, we briefly comment on the proposed formalisation.

The GRC system consists of one or more train trails which are traversed by a road. To avoid collisions between trains and cars a bar is automatically operated at the crossing. Let us call I the portion of train trails which crosses the road. To properly control the bar, sensors are placed



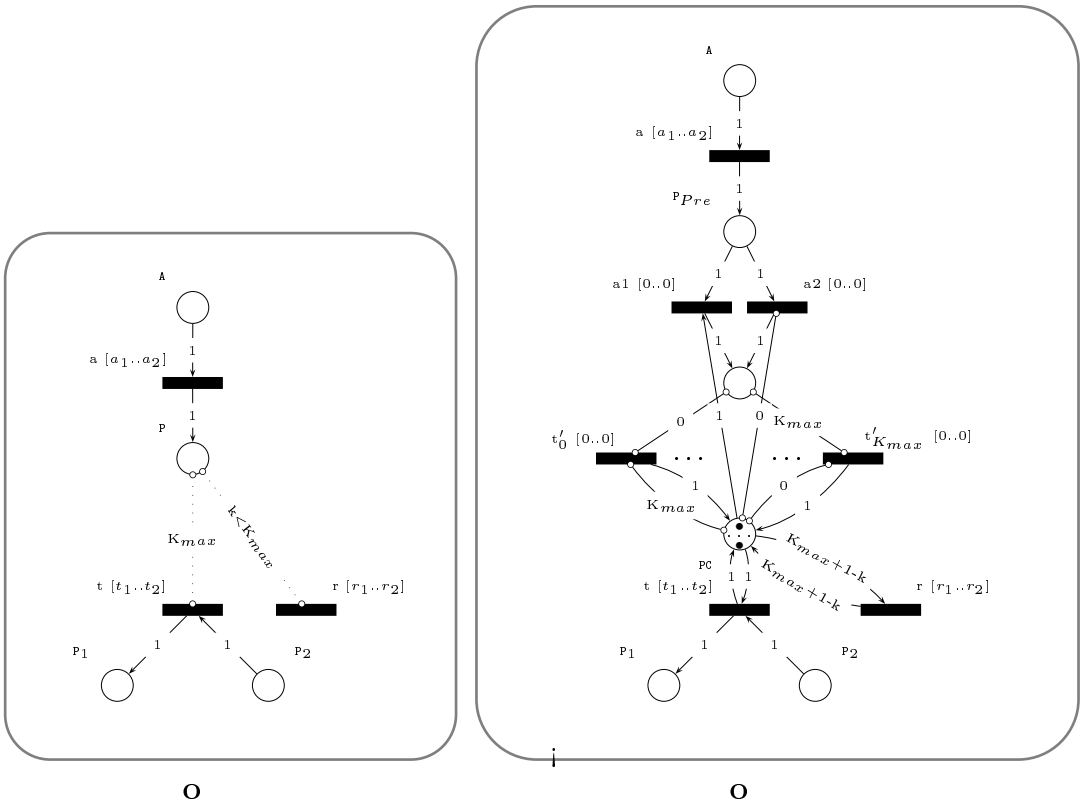
(a) SEM2 (b) Implementation of SEM2

Figure 12: Inhibitor Arcs and Time (3)

on the trails to detect the arrival and the exit of trains: the arrival of a train must be signalled somewhat in advance wrt the train entering region I, whereas the exit is signalled exactly when a train exits I. We call R the portion of train trails included between the place where entering sensors are placed and the beginning of I (see Figure 14). All trains have a minimum and a maximum speed so that they take a minimum and a maximum, yet finite and non-null, time to traverse R and I. The control of the bar operates as follows. Whenever a train enters R, this is detected and signalled by a sensor; similarly when a train exits I. If a train enters R and the bar is open (up), then a command is issued to the bar to close. This takes a fixed amount of time (γ). As soon as no more trains are in R or in I (this must be computed by the control apparatus on the basis of entering and exiting signals) the opening command is issued to the bar, which again takes γ time units to open (notice that, in this description, we assume that if a train enters R while the bar is opening, the control must wait until the bar is open before restarting closing it). The designer's job is to set system parameters (e.g., the length of R and the duration γ) in such a way that the following properties hold:

- Safety property: When a train is in I the bar is closed
- Utility property: the bar is closed only for the time that is strictly necessary to guarantee the safety property.

Let us now formalise the GRC system through real-time synchronised Petri nets. Figure 15 shows two objects: `Train` and `Level Crossing`. The `Train` object represents the entry and the exit of trains into a critical region: a train enters into the critical region with method `entry`, it stays first in the section corresponding to `R` (place `p2`), for a certain amount of time, represented by transition `in`. Then, it enters region `I`, represented by place `p3`, and finally it leaves the critical region with method `exit`. Several trains may be simultaneously in the critical region, however their entry is not simultaneous. Indeed method `entry` can fire more than once, since place `p1` contains always one token. However, method `entry` cannot fire twice or more simultaneously, and



(a) SEM2

(b) Implementation of SEM2

Figure 13: Inhibitor Arcs and Time (4)

there is a delay of at least t'_1 between two trains entering the critical region. The fact that two trains may be simultaneously in region `R` or in region `I` depends also of the values of $t_1, t_2, t'_1, t''_1, t''_2$. Indeed, if $t_2 + t''_2 < t'_1$ then there will be at most one train in the critical region.

The `Level Crossing` object represents the behaviour of the critical region, i.e., the level crossing. The level is up iff no train is currently in the critical region, or entering it.

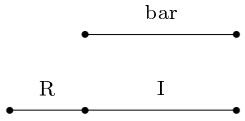


Figure 14: Critical Section

Each time a train enters the critical region, the `signal_entry` method fires. This is due to the fact that the `signal_entry` requires a synchronisation with method `entry` *and* the time interval of `signal_entry` is [0..0] (i.e., `signal_entry` *must* fire immediately when it is enabled). The `signal_entry` method increases the number of tokens in place `Counter`, whose role is to count the number of trains that are currently in the critical region. If the barrier is up and if a train arrives in the critical region, transition `go_down` fires and the barrier begins to go down (place `mv_down`). After a certain amount of time γ, represented by transition `end_down`, the barrier is finally down (place `down`).

Each time a train leaves the critical region, by activating method `exit`, the `signal_exit` method fires simultaneously. This method simply decreases by one the number of trains that are currently in the critical region. As soon as there are no more trains in the critical region, i.e. place `Counter` is empty, transition `go_up` fires (because of the inhibitor arc of weight 0, and time interval [0..0]

attached to go_up). The barrier begins to go up (place mv_up), and after a certain amount of time γ, represented by transition end_up, it arrives in the up position (place up).

When trains are in the critical region, and the barrier is already down, neither method go_down nor method go_up can fire.

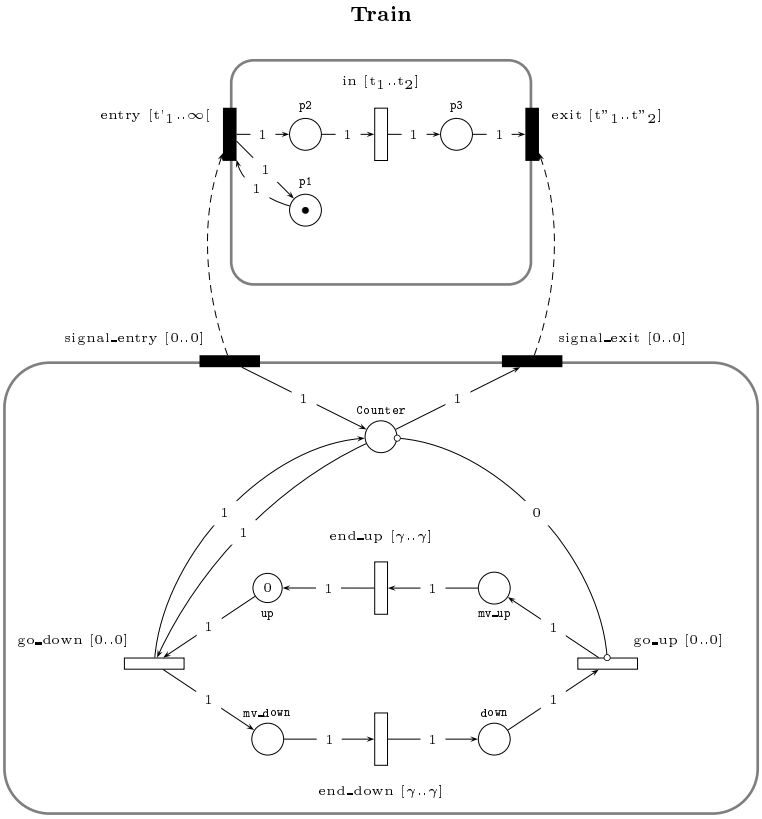


Figure 15: Train and Level Crossing

Remark 4.1 *The proposed example, though still rather simple, illustrates the suitability of the model for the description of -even complex- real-time systems. First, modularisation is naturally achieved through the definition of several objects and the use of the synchronisation mechanism to formalise their interaction.*

Second, the use of inhibitor arcs allows to achieve a good level of generality without resorting to more sophisticated models. In our case inhibitors arcs allow a natural formalisation of the counter mechanism which is essential for a proper description of the system.

Notice that pure Petri nets allow only the modelling of simplified versions of the GRC system (e.g. the case where only one train can traverse the regions R and I per time) whereas in order to deal with the general case more cumbersome formalisations are usually needed, all adopting Petri net-based models and other formalisms such as automata or process algebras [4]. It is an interesting exercise to pursue further generalisation of the example. For instance: modelling synchronisation among different trains besides synchronisation between trains and cars; allowing several trains to enter R simultaneously (allowing several train trails with place p1 containing more than one token); modelling a bounded number of train trails; etc. It is easy to realize that much generality can be achieved exploiting the "pure model" with anonymous tokens, up to some point where other typical features of the general CO-OPN formalism [2] must be included in the timed version. Once the system model has been built in terms of real-time synchronised Petri nets, we can use it to verify the desired properties. For instance we may build constraints on the values of train speed, length of R, and γ -which determine the value of interval $[t_1..t_2]$- in such a way that the safety and the utility property are guaranteed. Presently, such an analysis must be done informally through a typical simulation of the -timed- token game of the net. Further development of this research will build an axiom system in a suitable assertion language so that such an analysis can be carried over

as a formal proof, e.g., in the style of [3]. Also, once such an axiomatisation will be available, typical supporting theorem proving tools will be applicable to mechanise and making more robust system analysis [1].

5 Conclusion

This report presents *real-time synchronised Petri nets*, a class of high-level Petri nets (with inhibitor arcs, and synchronisation among Petri nets) with real-time constraints attached to transitions as relative time intervals. Strong time semantics has been defined for these nets: once it has been enabled, a transition *must* fire during the time interval attached to it.

Real-time synchronized Petri nets enable to easily specify the Generalised Railroad Crossing system (GRC). Thus, these nets promise to be a powerful tool for specifying complex critical systems.

Future works will concentrate on giving an axiomatisation to these nets in order to enable formal verification of logical properties.

References

[1] A. Alborghetti, A. Gargantini, and A. Morzenti. Providing automated support to deductive analysis of time critical systems. In *Sixth European Software Engineering Conference (ESEC'97)*, 1997.

[2] D. Buchs and N. Guelfi. A formal specification framework for object-oriented distributed systems. *IEEE Transactions on Software Engineering, Special Section on Formal Methods for Object Systems*, 26(7):635–652, July 2000.

[3] M. Felder, D. Mandrioli, and A. Morzenti. Proving properties of real-time systems through logical specifications and Petri net models. *IEEE Transactions on Software Engineering*, 20(2):127–141, February 1994.

[4] C. Heitmeyer and D. Mandrioli, editors. *Formal methods for real-time computing.* John Wiley & Sons, 1996.

[5] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods, and Practical Use*, volume 1 of *EATCS Monographs in Computer Science*. Springer-Verlag, 1992.

[6] W. Reisig. Petri nets and algebraic specifications. In *Theoretical Computer Science*, volume 80, pages 1–34. Elsevier, 1991.

[7] M. Wirsing. Algebraic specification. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Methods and Semantics, chapter 13, pages 675–788. North-Holland, Amsterdam, 1990.