SimHPN: A MATLAB Toolbox for Hybrid Petri Nets

- USER MANUAL v. 2.0-

Jorge Júlvez and Cristian Mahulea

Dept. of Computer Science and Systems Engineering
University of Zaragoza

April 2023

Contents

Section 1

Overview

Petri nets (PNs)[Mur89, DHP$^+$93] is a mathematical formalism for the description of discrete-event systems, that has been successfully used for modeling, analysis and synthesis purposes of such systems. A key feature of a PN is that its structure can capture graphically fundamental primitives in concurrency theory such as parallelism, synchronization, mutual exclusion, etc. The state of a PN system is given by a vector of non-negative integers representing the marking of its places.

As any other formalism for discrete event systems, PNs suffer from the *state explosion problem* which produces an exponential growth of the size of the state space with respect to the initial marking. One way to avoid the state explosion is to relax the integrality constraint in the firing of transitions and deal with transitions that are fired in real amounts. A transition whose firing amount is allowed to be any real number between zero and its enabling degree is said to be a *continuous transitions.* The firing of a continuous transition can produce a real, not integer, number of tokens in its input and output places. If all transitions of a net are continuous, then the net is said to be continuous. If a non-empty proper subset of transitions is continuous, then the net is said to be hybrid [DA10].

Different time interpretations can be considered for the firing of continuous transitions. The most popular ones are infinite and finite server semantics which represent a first order approximation of the firing frequency of discrete transitions. For a broad class of Petri nets, infinite server semantics offers a better approximation of the steady-state throughput than finite server semantics [MRS09]. Moreover, finite server semantics can be exactly mimicked by infinite server semantics in discrete transitions simply by adding a self-loop place. A third firing semantics, called product semantics, is also frequently used when dealing with biochemical and population dynamics systems.

SimHPN is a MATLAB embedded software that provides support for infinite server and product semantics in both, discrete and continuous, types of transition. A description of a preliminary version of this software can be found in [JM10, JMV11, JMV12]. This is the first MATLAB package that enables the analysis and simulation of hybrid nets with these two firing semantics. There already exists a toolbox dealing with discrete Petri nets [MMP03], and one for the so-called first order hybrid Petri nets [SGS08] which provides support for continuous transitions under finite server semantics. The main features of the *SimHPN* toolbox are:

1. simulation of hybrid Petri nets under different server semantics;
2. computation of steady state throughput bounds;
3. computation of minimal P-T *semiflows*;
4. optimal sensor placement;
5. optimal control algorithm;
6. import models from different graphical Petri net editors.

Figure 1.1 shows the Graphical user Interface of SimHPN automatically opened in MATLAB once

```
>> SimHPN
```

command is executed at the command prompt.

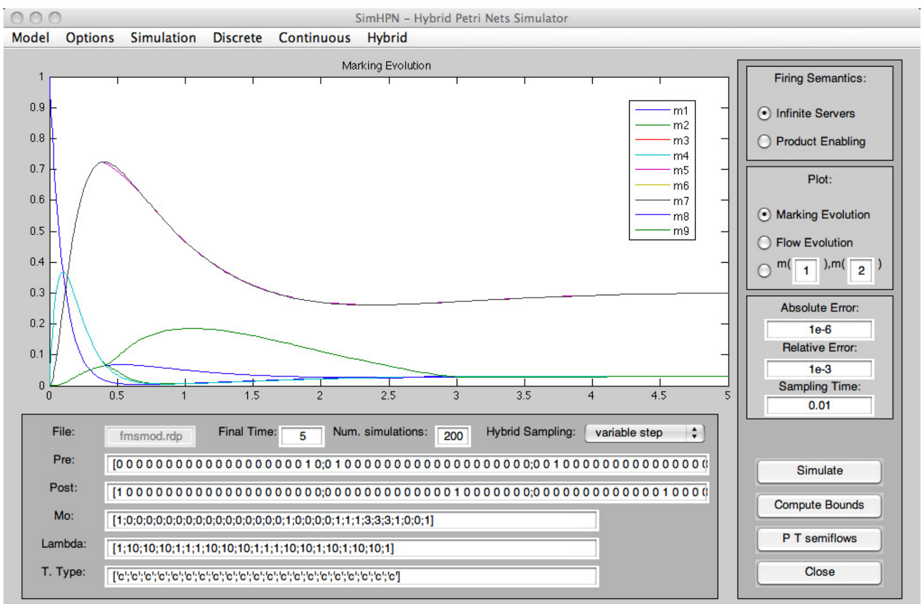


Figure 1.1: Sketch of the main window of SimHPN

Section 2

Installation Guide

The *SimHPN* toolbox has been developed in the GISED research group (Group of Discrete Event Systems Engineering, **URL:** `http://webdiis.unizar.es/GISED/`), University of Zaragoza, Spain. It offers a collection of tools devoted to simulation, analysis and synthesis of dynamical systems modeled by hybrid Petri nets. Its embedding in the MATLAB environment provides the considerable advantage of creating powerful algebraic, statistical and graphical instruments exploiting the high quality routines available in MATLAB.

SimHPN is a commercial software product for scientific purposes. Its demo version can be freely downloaded from:

URL: `http://webdiis.unizar.es/GISED/?q=tool/simhpn`

If you want to have access to the full version, please contact us at: `simhpn@unizar.es`

SimHPN runs on the MATLAB environment and the full version works, in principle, in any operating system that supports MATLAB R2008 or superior. Anyway, the demo version that can be downloaded from the previous link has been compiled on a particular version of MATLAB. If you experience problems in running it please contact us by email.

To install the toolbox, unzip the contents of the archive **SimHPN_P.zip** under a folder of your choice. This operation will create the directory *SimHPN* in the folder you chose. Launch the MATLAB, change the working directory to the folder *SimHPN* and type

```
>> SimHPN
```

at the MATLAB prompt. The *SimHPN* GUI will become operational.

Section 3

Petri nets overview

Hybrid Petri nets [DA10, BMG00] represent a powerful modeling formalism that allows the integration of both continuous and discrete dynamics in a single net model. This section defines the class of hybrid nets supported by $SimHPN$. In the following, the reader is assumed to be familiar with Petri nets (PNs) (see [Mur89, DHP$^+$93] for a gentle introduction).

3.1 Untimed Hybrid Petri net systems

Definition 3.1. A *Hybrid Petri Net (HPN) system* is a pair $\langle \mathcal{N}, \boldsymbol{m_0} \rangle$, where: $\mathcal{N} = \langle P, T, \boldsymbol{Pre}, \boldsymbol{Post} \rangle$ is a *net structure*, with set of places P, set of transitions T, pre and post incidence matrices $\boldsymbol{Pre}, \boldsymbol{Post} \in \mathbb{R}_{\geq 0}^{|P|\times|T|}$, and $\boldsymbol{m}_0 \in \mathbb{R}_{\geq 0}^{|P|}$ is the *initial marking.*

The token load of the place p_i at marking $\boldsymbol{m}$ is denoted by m_i and the *preset* and *postset* of a node $X \in P \cup T$ are denoted by ${}^{\bullet}X$ and $X^{\bullet}$, respectively. For a given incidence matrix, e.g., $\boldsymbol{Pre}$, $\boldsymbol{Pre}(p_i, t_j)$ denotes the element of $\boldsymbol{Pre}$ in row i and column j.

In a HPN, the set of transitions T is partitioned in two sets $T = T^c \cup T^d$, where T^c contains the set of continuous transitions and T^d the set of discrete transitions. In contrast to other works, the set of places P is not explicitly partitioned, i.e., the marking of a place is a natural or real number depending on the firings of its input and output transitions. Nevertheless, in order to make net models easier to understand, those places whose marking can be a real non-integer number will be depicted as double circles (see p_1^1 in Fig. 3.1), and the rest of places will be depicted as simple circles (such places will have integer markings, see p_5^1 in Fig. 3.1). Continuous transitions are graphically depicted as two bars (see t_4^1 in Fig. 3.1), while discrete transitions are represented as empty bars (see t_5^1 in Fig. 3.1), .

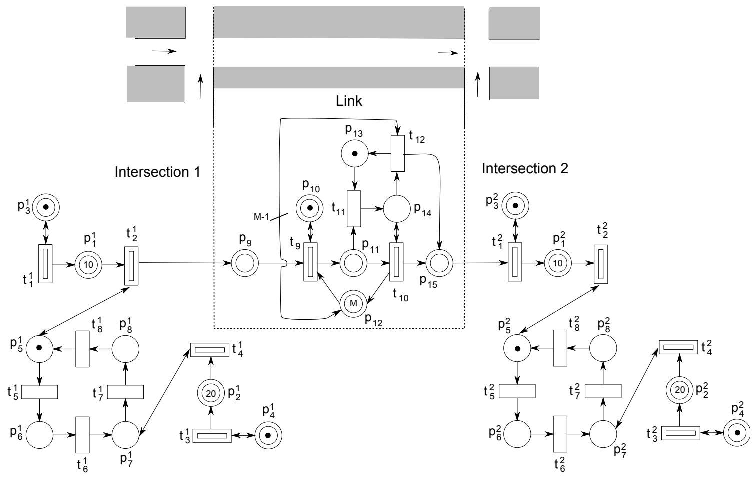


Figure 3.1: HPN model of 2 intersections connected by a link.

Two enabled transitions t_i and t_j are in conflict when they cannot occur at the same time. For this, it is necessary that ${}^\bullet t_i \cap {}^\bullet t_j \neq \emptyset$, and in that case it is said that t_i and t_j are in structural conflict relation. Right and left non negative annullers of the token flow matrix $\boldsymbol{C}$ are called T- and P-*semiflows*, respectively. A semiflow $\mathbf{v}$ is *minimal* when its support, $\|\mathbf{v}\| = \{i \mid \mathbf{v}(i) \neq 0\}$, is not a proper superset of the support of any other semiflow, and the greatest common divisor of its elements is one. If there exists $\boldsymbol{y} > 0$ such that $\boldsymbol{y} \cdot \boldsymbol{C} = 0$, the net is said to be *conservative*, and if there exists $\boldsymbol{x} > 0$ satisfying $\boldsymbol{C} \cdot \boldsymbol{x} = 0$, the net is said to be *consistent*. As it will be seen, the basic tasks that $SimHPN$ can perform on untimed hybrid Petri nets are related to the computation of minimal T- and P-semiflows.

The enabling degree of a continuous transition $t_j \in T$ is:

$$enab(t_j, \boldsymbol{m}) = \begin{cases} \min\limits_{p_i \in {}^\bullet t_j} \left\lfloor \dfrac{m_i}{\boldsymbol{Pre}(p_i, t_j)} \right\rfloor & \text{if } t_j \in T^d \\ \min\limits_{p_i \in {}^\bullet t_j} \dfrac{m_i}{\boldsymbol{Pre}(p_i, t_j)} & \text{if } t_j \in T^c \end{cases} \tag{3.1}$$

Transition $t_j \in T$ is *enabled* at $\boldsymbol{m}$ iff $enab(t_j, \boldsymbol{m}) > 0$. An enabled transition $t_j \in T$ can fire in any amount α such that $0 \leq \alpha \leq enab(t_j, \boldsymbol{m})$ and $\alpha \in \mathbb{N}$ if $t_j \in T^d$ and $\alpha \in \mathbb{R}$ if $t_j \in T^c$. Such a firing leads to a new marking $\boldsymbol{m'} = \boldsymbol{m} + \alpha \cdot \boldsymbol{C}(\cdot, t_j)$, where $\boldsymbol{C} = \boldsymbol{Post} - \boldsymbol{Pre}$ is the token-flow matrix and $\boldsymbol{C}(\cdot, t_j)$ is its j column. If $\boldsymbol{m}$ is reachable from $\boldsymbol{m}_0$ through a finite sequence σ, the *state (or fundamental) equation*, $\boldsymbol{m} = \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma}$ is satisfied, where $\boldsymbol{\sigma} \in \mathbb{R}_{\geq 0}^{|T|}$ is the firing count vector. According to this firing rule the class of nets defined in Def 3.1 is equivalent to the class of nets defined in [DA10, BMG00].

3.2 Timed Hybrid Petri net systems

Different time interpretations can be associated to the firing of transitions. Once an interpretation is chosen, the state equation can be used to show the dependency of the marking on time, i.e., $\boldsymbol{m}(\tau) = \boldsymbol{m_0} + \boldsymbol{C} \cdot \boldsymbol{\sigma}(\tau)$. The term $\boldsymbol{\sigma}(\tau)$ is the firing count vector at time τ. Depending on the chosen time interpretation, the firing count vector $\sigma_j(\tau)$ of a transition $t_j \in T^c$ is differentiable with respect to time, and its derivative $f_j(\tau) = \dot{\sigma_j}(\tau)$ represents the *continuous flow* of t_j. As for the timing of discrete transitions, several definitions exist for the flow of continuous transitions. $SimHPN$ accounts for infinite server and product server semantics in both continuous and discrete transitions, and additionally, discrete transitions are also allow to have deterministic delays.

Definition 3.2. A *Timed Hybrid Petri Net (THPN) system* is a tuple $\langle \mathcal{N}, \boldsymbol{m}_0, Type, \boldsymbol{\lambda} \rangle$ where $\langle \mathcal{N}, \boldsymbol{m}_0 \rangle$ is a HPN, $Type : T \rightarrow \{id, pd, dd, ic, pc\}$ establishes the time semantics of transitions and $\boldsymbol{\lambda} : T \rightarrow \mathbb{R}_{\geq 0}$ associates a real parameter to each transition related to its semantics.

Any of the following semantics is allowed for a discrete transition $t_i \in T^d$:

- *Infinite server semantics* ($Type(t_i) = id$): Under infinite server semantics, the time delay of a transition t_i, at a given marking $\boldsymbol{m}$, is an exponentially distributed random variable with parameter $\lambda_i \cdot enab(t_i, \boldsymbol{m})$, where the integer enabling $enab(t_i, \boldsymbol{m})$ represents the number of active servers of t_i at marking $\boldsymbol{m}$.

- *Product server semantics* ($Type(t_i) = pd$): Under product server semantics, the time delay of a transition t_i at $\boldsymbol{m}$ is an exponentially distributed random variable with parameter $\lambda_i \cdot \prod_{p_j \in {}^\bullet t_i} \left\lfloor \frac{\boldsymbol{m}(p_j)}{\boldsymbol{Pre}(p_j, t_i)} \right\rfloor$, where $\prod_{p_j \in {}^\bullet t_i} \left\lfloor \frac{\boldsymbol{m}(p_j)}{\boldsymbol{Pre}(p_j, t_i)} \right\rfloor$ is the number of active servers.

- *Deterministic delay* ($Type(t_i) = dd$): A transition t_i with deterministic delay is scheduled to fire $1/\lambda_i$ time units after it became enabled.

Conflict resolution: When several discrete exponential transitions, under either infinite or product server semantics, are in conflict, a racing policy is adopted, i.e., the one with smaller time delay will fire first.

If a discrete transition with deterministic delay is not in conflict with other transitions, it is fired as scheduled, if it is in conflict then it is fired only if its schedule firing time is less than the firing time of the conflicting transition. The transition to fire, in the case of several conflicting deterministic transitions with same scheduled firing instance, is chosen probabilistically assigning the same probability to each conflicting transition. Furthermore after the firing of a deterministic transition, the timers of all the transitions in the same conflict are discarded.

For a continuous transition $t_i \in T^c$ the following semantics are allowed:

- *Infinite server semantics* ($Type(t_i) = ic$): Under *infinite server* the flow of a transition t_i is:

$$f_i = \lambda_i \cdot enab(t_i, \boldsymbol{m}) = \lambda_i \cdot \min_{p_j \in {}^\bullet t_i} \left\{ \frac{m_j}{\boldsymbol{Pre}(p_j, t_i)} \right\} \tag{3.2}$$

  Such an expression for the flow is obtained from a first order approximation of the discrete case [SR02] and corresponds to the *variable speed* of ([AD98])

- *Product server semantics* ($Type(t_i) = pc$): In a similar way to discrete transitions, the continuous flow under product server semantics is given by:

$$f_i = \lambda_i \cdot \prod_{p_j \ \in \ {}^\bullet t_i} \left\{ \frac{m_j}{\boldsymbol{Pre}(p_j, t_i)} \right\}$$

The described supported semantics cover the modeling of a large variety of actions usually associated to transitions. For instance, infinite server semantics, which are more general than finite server semantics, are well suited for modeling actions in manufacturing, transportation and logistic systems [DA10]; product server semantics are specially useful to modeling population dynamics [SR00] and biochemical reactions [HGD08]; and deterministic delays allow one to represent pure delays and clocks that appear, for instance, when modeling traffic lights in automotive traffic systems [VSBS10].

Section 4

Using SimHPN

4.1 Short description of the GUI

The $SimHPN$ simulator supports infinite server and product server semantics for both discrete and continuous transitions. Moreover, deterministic delays with single server semantics are also supported for discrete transitions. Both the data related to the model description, i.e., net structure, initial marking and timing parameter, and the output results, i.e., time trajectories, are MATLAB variables. At the end of the simulation, the user can export the data to the MATLAB workspace where can be used for further analysis.

The $SimHPN$ toolbox (`http://webdiis.unizar.es/GISED/?q=tool/simhpn`) provides a Graphical User Interface (GUI) that enables the user to easily perform simulations and carry out analysis methods. This GUI consists of a MATLAB figure window, exhibiting a *Menu bar* and three control panels:

(i) *Drawing Area*,

(ii) *Options panel*, and

(iii) *Model Management panel*.

Fig. 4.1 presents a hard-copy screenshot of the main window opened by $SimHPN$ toolbox, where all the component parts of the GUI are visible.

The *Menu bar* (placed horizontally, on the top of the window in Fig. 4.1) displays a set of six drop-down menus at the top of the window, where the user can select different features available in the $SimHPN$ toolbox. These menus are: *Model*, *Options*, *Simulation*, *Structural*, *Continuous* and *Hybrid*.

The *Model menu* contains the pop-up menus:

- *Import from Pmeditor*,

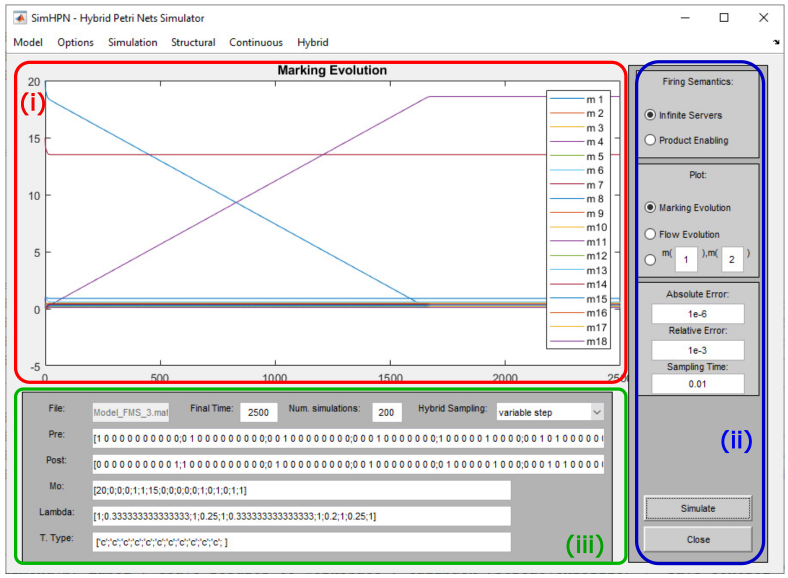


Figure 4.1: Sketch of the main window of $SimHPN$

- *Import from TimeNet* and
- *Import from .mat file*
- *Save to .mat file*

that implement several importing options for the matrices, $\boldsymbol{Pre}$, $\boldsymbol{Post}$, $\boldsymbol{m}_0$, etc, that describe the net system. Such matrices can be introduced manually or through two Petri nets editors: PMEditeur and TimeNet [ZK95]. Moreover, the matrices can be automatically loaded from a *.mat* file (MATLAB file format) or loaded from variables defined in the workspace, this is done just by writing the name of the variable to be used in the corresponding edit boxes.

If the model is imported from a *.mat* file, the following variables should be defined in the *.mat* file:

- *Pre* - is a $|P|\times\ |T|$ matrix containing the weight of the arcs from places to transitions;
- *Post* - is a $|P|\times\ |T|$ containing the weight of the arcs from transitions to places;
- $M0$ - is a column vector of dimension $|P|$ containing the initial marking;
- *Lambda* - is a column vector of dimension $|T|$ containing the firing rates of transitions:

- $Trans\_Type$ - is a column vector of dimension $|T|$ containing the type of transitions. Each element can have one of the following values:
  - $'c'$ - for continuous transition;
  - $'d'$ - for stochastic discrete transitions;
  - $'q'$ - for deterministic discrete transitions.

The last option (i.e., *Save to .mat file*) saves to a .mat file the matrices already defined in the edit boxes of $SimHPN$.

The *Options* menu contains only the pop-up menu

- *Show Figure Toolbar*

that allows to show the characteristic toolbar of the MATAB figure object that permits, for example, the use of zoom tool on the displayed graphic in the *Drawing Area*.

The *Simulation menu* contains the pop-up menus:

- *Markings to plot*,
- *Flows to plot*, and
- *Save results to workspace*.

The pop-up menus *Markings to plot*, *Flows to plot* allow the user to select the components of marking vector and flow vector that will be plotted after simulation in the *Drawing area*. The pop-up menu *Save results to workspace* permits to export, after simulation, the marking and flow evolution to variables in the MATLAB workspace.

The rest of the menus call procedures for analysis and synthesis of PN model if it is assumed discrete Petri net (*Structural* menu), fully continuous Petri net (*Continuous* menu), and Hybrid Petri net (*Hybrid* menu). The different functions of these menus will be further explained in the following sections.

The *Drawing area* (located in the left and central side of the window in Fig. 4.1), is a MATLAB axes object where the trajectories of the simulation results are plotted. The components of markings and flows that will be represented are selected from the menu.

The *Options panel* (placed, as a horizontal bar, on the right part of the window Fig. 4.1) presents a number of options related to the model. From top to bottom: (a) two radio buttons to select the firing semantics for continuous and discrete exponential transitions; (b) three radio buttons allowing to select the variables to be plotted in the *Drawing Area*, the simulator allows one to plot the evolution of the marking of the places, the evolution of the flow of the transitions and the evolution of the marking of one place vs. the marking of other place;

(c) three edit boxes to fix the maximum absolute and relative errors allowed by the simulated trajectory and the sampling time used in simulations (see next subsection for more details on the selection of the sampling time); (d) a *Simulate* button to start a new simulation; (e) a *Compute Bounds* button that computes performance bounds for continuous nets under infinite server semantics; (f) a *P T semiflows* button to compute the minimal P- and T-semiflows of the net, the results are displayed on the MATLAB command window and can be used for future analysis tasks; and (g) a *Close* button to close the *SimHPN* toolbox.

The *Model Management Panel* panel is composed of different edit boxes (placed in the bottom left corner of the window in Fig. 4.1), where the *SimHPN* toolbox displays the current values of the matrices describing the net system and permits to select the simulation time and the number of simulations to be performed (this last parameter is ignored if the net contains no stochastic transitions). The required matrices for a system in order to be simulated are: $\boldsymbol{Pre}$ and $\boldsymbol{Post}$ matrices, initial marking $\boldsymbol{m}_0$, the parameter $\boldsymbol{\lambda}$ of each transition, and the type of each transition. This last parameter is equal to 'c' for continuous transitions, to 'd' for stochastic discrete transitions and to 'q' for deterministic discrete transitions. Notice that if the type of a transition is 'q' then single server semantics is adopted for its firing and therefore the selection of firing semantics in the *Options panel* will be ignored for this transition.

4.2 Analysis and design tools

In the following, we describe the different analysis and design tools implemented in SimHPN.

The *Structural menu* contains different analysis techniques to compute the following properties of a given net structure:

- *Conservativeness*: Test if the net structure is conservative and computes its different conservative components.
- *Consistency*: Test if the net structure is consistent and computes its different repetitive components.
- *P T semiflows*: Compute the T-semiflows and P-semiflows of the net structure, and gives you the option to save the solutions to the workspace.

The *Continuous menu* contains different analysis and design tools developed for continuous and timed continuous PN systems. In the following, the different implemented methods are detailed.

The *Compute bounds* menu provide algorithms that compute lower and upper bound for infinite servers semantics and upper bound for finite servers semantics for the case of *unforced* systems (with no control). Based on the methods developped in [?]. It is only valid for mono-t-semiflow systems.

The *Optimal* submenu contains the pop-up menus *Optimal Observability* and *Optimal Control*. Such pop-up menus perform calls to the algorithms for computing optimal steady state for the *forced* case (with control) [?] and optimal sensor placement for continuous Petri nets with infinite server semantics.

There are different submenus that contain different algorithms for the implementation of *control laws*, permitting the system to reach a desired final marking if the net is continuous under infinite server semantics.

After using a particular control law, it is possible to use the menu *Save results to workspace* to save the computed input and the marking trajectory of the forced system. Between the implemented algorithms are different ON/OFF Methods [?], Affine control laws [?], minimum-time control law [?], [?], minimum-time control flow [?]. See the cited references for the particular pre-requisites of each method.

The menus for Structural controllability analysis and Diagnosis are explained with more detail in the following sections.

4.2.1 Structural controllability analysis

This submenu provides a set of algorithms that have been specifically implemented for the structural controllability analysis of timed continuous Petri nets [?]. These algorithms offer various tests for determining the net rank-controllability property of a given TCPN and the computation of the sets of equilibrium markings of the system.

In a live and bounded TCPN system, the concept of net rank-controllability serves as a sufficient condition for determining whether the system is controllable over its connected equilibrium sets. See Section 5.4 for examples and more details on the application of these tools.

Influence of controllable transitions

This routine computes the sets of *influenced places and transitions by the controllable transitions*, P_I and T_I. This is a necessary condition for net rank-controllability and it indicates the sets of nodes whose marking and flow can be affected by the activity on the controllable transitions, regardless of the configuration. If a controllable transition does not influence all places in the net, the net cannot be net rank-controllable. It is carried out by implementing Algorithm 1 in [?].

When the user selects this routine, a window is opened and the user should introduce the set of controllable transitions. After this, the results are shown in a pop-up window containing one of two possible messages: 1) A message indicating that influence is not total and indicating the sets of influenced nodes, $P_I \subset P$ and $T_I \subset T$, 2) A message indicating that influence is total, i.e., $P_I = P$ and $T_I = T$.

Net rank-controllability test

This routine verifies sufficient conditions for net rank-controllability. It is carried out by implementing Algorithm 2 in [**?**]. It is efficiently performed by verifying some structural conditions:
• **Structural Condition 1 (SC1)**: Given $\mathcal{N}$, the influence of T_c is total.
• **Structural Condition 2 (SC2)**: Given $\mathcal{N}$, for all choice place, $p_c \in \{p \in P || p^\bullet| > 1\}$, $p_c^\bullet \subseteq T_c$.
• **Structural Condition 3 (SC3)**: Given $\mathcal{N}$, let $T_F = \{t \in T || t^\bullet| > 1\}$ be the set of fork transitions. For all $t_f \in T_F$, we assume that $|\{t_f^\bullet\}^\bullet \cap T_{nc}| \leq 1$.

When the user selects this routine, a window is opened and the user should introduce the set of controllable transitions. After this, the results are shown in a pop-up window indicating if the net fulfills the property. In case it is not fulfilled, it indicates which one of the structural conditions does not hold.

Equilibrium connectivity graph

This routine computes the equilibrium sets and their connectivity for a specified TCPN system. The output is a graph where each node represents a configuration of the system whose region contains equilibrium markings, and the edges indicate the connection between the equilibrium markings of the different regions. You can save the results of this analysis by selecting the "Save Equilibrium Connectivity Graph to Workspace" submenu. Additionally, if all the equilibrium sets are connected, the function will return a message indicating this.

4.2.2 Diagnosis

The *Diagnosis* submenu calls the algorithms that perform fault diagnosis for untimed continuous Petri nets [MSCS12]. Once the user selects this option, a window as in Fig. 4.2 is opened and the user should introduce the input parameters:

1. The number of observable transitions. Without loss of generality, it is assumed that the first transitions are observable and the rest are unobservable.

2. The firing sequence of observable transitions.

3. The firing amount of each transition of the firing sequence.

4. Number of fault classes.

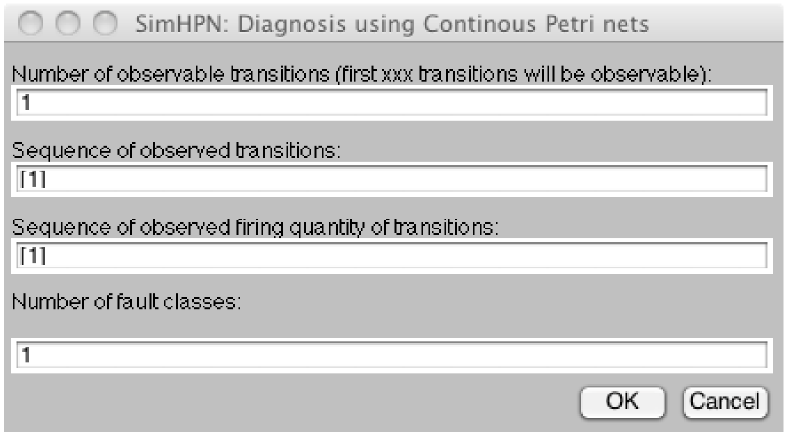


Figure 4.2: Sketch of the uicontrols for the diagnosis.

After this, an uicontrol will be used to introduce the transitions belonging to each fault class. All the results are shown at the MATLAB command prompt. See Section 5.4 for examples and more details on the parameters for fault detection.

4.3 Script simulation

SimHPN permits the simulation of a hybrid Petri net from the MATLAB prompt, without open the GUI. This is useful if one wants to implement a script and simulate a hybrid Petri net for some particular input parameters. This can be done in two different ways depending on the type of the input model.

Continuous Petri net. If the net has only continuous transitions, the following command can be used for its simulation:

```
[m,f,t] = SimHPN_Simul(Pre, Post, m0, tf, lambda, seman, erel, eabs);
```

where,

- *Pre* - is a $|P|\times\ |T|$ matrix containing the weight of the arcs from places to transitions;
- *Post* - is a $|P|\times\ |T|$ containing the weight of the arcs from transitions to places;
- $m0$ - is a column vector of dimension $|P|$ containing the initial marking;

- tf - is the final time for simulation;
- *lambda* - is a column vector of dimension $|T|$ containing the firing rates of transitions:
- $seman = 1$ - if transitions are with infinite server semantics and $seman = 2$ - if product firing semantics is considered for transitions;
- $erel$ - relative error;
- $eabs$ - absolute error.

The returned parameters are:

- m - a matrix of dimension $|t| \times |P|$ is the matrix defining the marking evolutions;
- f - a matrix of dimension $|t| \times |T|$ is the matrix defining the flow evolutions;
- t - a row vector containing the time instants for which the markings and the flows are defined.

Hybrid Petri net. If the net is not purely continuous, function *SimHPN_SimHyb* should be used. This function is defined as follows:

```
[m,f,t] = SimHPN_SimHyb(Pre,Post,lambda,type,m0,nsim,tsim,seman,mode,delta)
```

The input parameters are:

- Pre - is a $|P|\times$ $|T|$ matrix containing the weight of the arcs from places to transitions;
- $Post$ - is a $|P|\times$ $|T|$ containing the weight of the arcs from transitions to places;
- *lambda* - is a column vector of dimension $|T|$ containing the firing rates of transitions:
- *type* - is a column vector of dimension $|T|$ containing the type of transitions. Each element can have one of the following values:
  - $'c'$ - for continuous transition;
  - $'d'$ - for stochastic discrete transitions;
  - $'q'$ - for deterministic discrete transitions.
- $m0$ - is a column vector of dimension $|P|$ containing the initial marking;
- $nsim$ - number of simulations especially useful for stochastic transitions;

- *tsim* - simulation time
- $seman = 1$ - if continuous transitions are with infinite server semantics and $seman = 2$ is product firing semantics is considered for continuous transitions;
- $mode = 1$ - if constant hybrid sampling step will be used; $mode = 2$ - if variable hybrid sampling step will be used;
- *delta* - sampling timed used in the case of $mode = 1$.

After a simulation, the returned parameters are:

- m - a matrix of dimension $|t| \times |P|$ is the matrix defining the marking evolutions;
- f - a matrix of dimension $|t| \times |T|$ is the matrix defining the flow evolutions;
- t - a row vector containing the time instants for which the markings and the flows are defined.

Section 5

Examples of utilization

This section presents some systems that are analyzed and simulated using $SimHPN$.

5.1 An assembly line

The Petri net system in Fig. 5.1 represents an assembly line with kanban strategy (see [ZRS01]). The system has two stages that are connected by transition t_{14}. The first stage is composed of three lines (starting from p_2, p_3 and p_4 respectively) and three machines (p_{23}, p_{24} and p_{25}). Places p_{26}, p_{27} and p_{28} are buffers at the end of the lines. The second stage has two lines that require the same machine/resource p_{18}. The number of kanban cards is given by the marking of places p_2, p_3 and p_4 for the first stage, and by the marking of p_{32} for the second stage. The system demand is given by the marking of p_1. We will make use of this net system to illustrate some of the features of $SimHPN$.

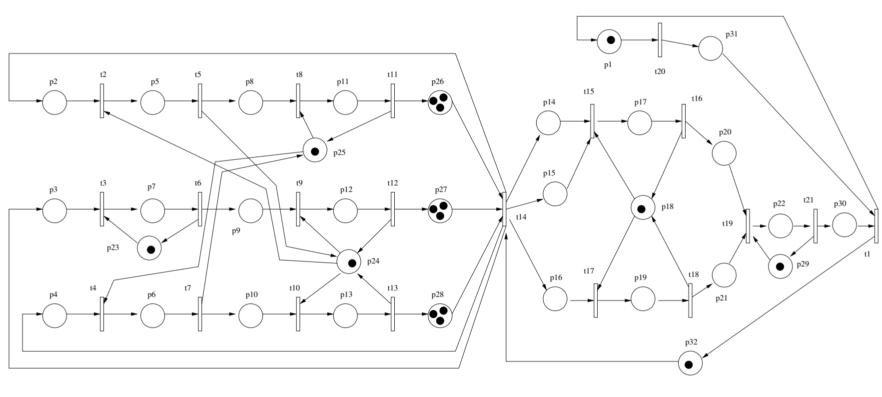


Figure 5.1: An assembly line with kanban strategy.

Given that all transitions represent actions that can potentially have high working loads, all transitions are considered continuous. Moreover, infinite server semantics will be adopted for all of them. Let the initial marking be $\boldsymbol{m}_0(p_1) = \boldsymbol{m}_0(p_{18}) = \boldsymbol{m}_0(p_{23}) = \boldsymbol{m}_0(p_{24}) = \boldsymbol{m}_0(p_{25}) = \boldsymbol{m}_0(p_{29}) = \boldsymbol{m}_0(p_{32}) = 1$, $\boldsymbol{m}_0(p_{26}) = \boldsymbol{m}_0(p_{27}) = \boldsymbol{m}_0(p_{28}) = 3$ and the marking of the rest of places be equal to zero. Let us assume that the firing rates of the transitions are $\boldsymbol{\lambda}(t_2) = \boldsymbol{\lambda}(t_3) = \boldsymbol{\lambda}(t_4) = \boldsymbol{\lambda}(t_8) = \boldsymbol{\lambda}(t_9) = \boldsymbol{\lambda}(t_{10}) = \boldsymbol{\lambda}(t_{14}) = \boldsymbol{\lambda}(t_{15}) = \boldsymbol{\lambda}(t_{17}) = \boldsymbol{\lambda}(t_{19}) = \boldsymbol{\lambda}(t_{20}) = 10$, $\boldsymbol{\lambda}(t_1) = \boldsymbol{\lambda}(t_5) = \boldsymbol{\lambda}(t_6) = \boldsymbol{\lambda}(t_7) = \boldsymbol{\lambda}(t_{11}) = \boldsymbol{\lambda}(t_{12}) = \boldsymbol{\lambda}(t_{13}) = \boldsymbol{\lambda}(t_{16}) = \boldsymbol{\lambda}(t_{18}) = \boldsymbol{\lambda}(t_{21}) = 1$.

Computation of minimal P-T *semiflows*: *SimHPN* implements the algorithm proposed in [Sil85] to compute the minimal P-*semiflows* of a Petri net. The same algorithm applied on the transpose of the incidence matrix yields the minimal T-*semiflows* of the net. Notice that P and T-*semiflows* just depend on the structure of the net and not on the continuous or discrete nature of the transitions. The result of applying the algorithm on the net in Fig. 5.1 is the set of 12 minimal P-semiflows that cover every place, i.e., it is conservative, and the set that contains the only minimal T-semiflow which is a vector of ones, i.e., it is consistent.

Throughput bounds: When all transitions are continuous and work under infinite server semantics, the following programming problem can be used to compute an upper bound for the throughput, i.e., flow, of a transition ([JRS05]):

$$\begin{aligned}
&\max\{\phi_j \mid \boldsymbol{\mu}^{ss} = \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma}, \\
&\quad \phi_j^{ss} = \lambda_j \cdot \min_{p_i \in {}^\bullet t_j} \left\{ \tfrac{\mu_i^{ss}}{Pre(p_i,t_j)} \right\}, \forall t_j \in T, \\
&\quad \boldsymbol{C} \cdot \phi^{ss} = \mathbf{0}, \\
&\quad \boldsymbol{\mu} ss, \boldsymbol{\sigma} \geq \mathbf{0}\}.
\end{aligned} \tag{5.1}$$

This non-linear programming problem is difficult to solve due to the minimum operator. When a transition t_j has a single input place, the equation reduces to (5.2). And when t_j has more than an input place, it can be relaxed (linearized) as (5.3).

$$\phi_j^{ss} = \lambda_j \cdot \frac{\mu_i^{ss}}{Pre(p_i, t_j)}, \text{if } p_i = {}^\bullet t_j \tag{5.2}$$

$$\phi_j^{ss} \leq \lambda_j \cdot \frac{\mu_i^{ss}}{Pre(p_i, t_j)}, \forall p_i \in {}^\bullet t_j, \text{otherwise} \tag{5.3}$$

This way we have a single linear programming problem, that can be solved in polynomial time. Unfortunately, this LPP provides in general a non-tight bound, i.e., the solution may be non-reachable for any distribution of the tokens verifying the P-*semiflow* load conditions, $\boldsymbol{y} \cdot \boldsymbol{m}_0$. One way to improve this bound is to force the equality for at least one place per synchronization (a transition with more than one input place). The problem is that there is no way to know in advance which of the input places should restrict the flow. In order to overcome this problem, a branch & bound algorithm can be used to compute a reachable steady state marking.

SimHPN implements such a branch & bound algorithm to compute upper throughput bounds of continuous nets under infinite server semantics. For the system in Fig. 5.1 with the mentioned $\boldsymbol{m}_0$ and $\boldsymbol{\lambda}$ the obtained throughput bound for t_1 is 0.3030. Given that the only T-semiflow of the net is a vector of ones, this value applies as an upper bound for the rest of transitions of the net.

Optimal Sensor Placement:

Assuming that each place p can be measured at a different cost $c(p) > 0$ the optimal sensor placement problem of continuous nets under infinite server semantics is to decide the set of places $P_o \subseteq P$ to be measured such that the net system is observable at minimum cost. This problem can be seen as a Set Covering Problem which is NP-hard in the strong sense [GJ79]. For a set of places P_o, let K_{P_o} be the set of observable places. It will be said that P_o is covering K_{P_o}. The problem is to determine a set P_{o_i} with minimum cost such that the covered elements contain all the places of the net.

Considering that n is the number of places, the brute force approach to solve this problem is to try all subsets of places of size n, $n-1$, $\cdots$, 1. From those subsets ensuring the observability of the continuous Petri net system, the one with minimum cost is taken. In order to reduce the number of the subsets, some graph-based properties can be used. The idea is to group the set of places in equivalence classes such that only one place per class can belong to the optimal solution. These equivalence classes are called *threads* and are the places belonging to the maximally connected subnets finishing in an attribution place (place with more than one input transitions) or in a join (transition with more than one input place). Initially, all places of the net belong only to one thread. The following algorithm can be used to reduce the number of elements from threads.

These reductions preserve the optimality of the solution and the covering problem can be started using the resulted threads. It is necessary to generate all combinations taking at most one place from each thread and then check the observability of the system. If the system is observable, the solution is kept if has a cost lower than the candidate solution. A good choice is starting with the first places of each thread and going backward since the following property is true: if the system is not observable for the current set of measured places, it is not necessary to advance in the threads because the system will not be observable.

The algorithm is still exponential but the structural properties presented can reduce drastically the number of observability checks. For the continuous nets considered in this section, measuring all input places in join transitions the net system is observable. This is also the solution of the optimal sensor placement problem for any cost associated to transitions.

Optimal Steady-State: The only action that can be performed on a continuous Petri nets is to slow down the flow of its transitions. If a transition can be controlled (its flow reduced or even stopped), we will say that is a *controllable* transition. The forced flow of a controllable transition t_j becomes $f_j - u_j$, where f_j is the flow of the unforced system, i.e. without control, and u_j is the control action $0 \leq u_j \leq f_j$.

In production control is frequent the case that the profit function depends on production (benefits in selling), working process and amortization of investments. Under linear hypothesis for fixed machines, i.e., $\boldsymbol{\lambda}$ defined, the profit function may have the following form:

$$\mathbf{w}^T \cdot \boldsymbol{f} - \boldsymbol{z}^T \cdot \boldsymbol{m} - \boldsymbol{q}^T \cdot \boldsymbol{m}_0 \tag{5.4}$$

where $\boldsymbol{f}$ is the throughput vector, $\boldsymbol{m}$ the average marking, $\boldsymbol{w}$ a gain vector w.r.t. flows, $\boldsymbol{z}^T$ is the cost vector due to immobilization to maintain the production flow and $\boldsymbol{q}^T$ represents depreciations or amortization of the initial investments.

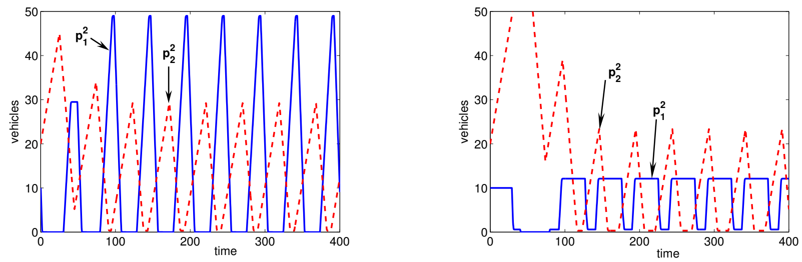
The algorithm used to compute the optimal steady state flow (and marking) is very much alike the one used to compute the performance bounds, with the difference that the linear programming problem that needs to be solved is:

$$\begin{cases} \max\{\boldsymbol{w}^T \cdot \boldsymbol{f} - \boldsymbol{z}^T \cdot \boldsymbol{m} - \boldsymbol{q}^T \cdot \boldsymbol{m}_0 \mid \boldsymbol{C} \cdot \boldsymbol{f} = 0, \\ \qquad \boldsymbol{m} = \boldsymbol{m}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma}, \\ \qquad f_j = \lambda_j \cdot \left(\frac{m_i}{Pre(p_i, t_j)} \right) - v(p_i, t_j), \\ \qquad\qquad \forall p_i \in {}^{\bullet}t_j, v(p_i, t_j) \geq 0 \\ \qquad \boldsymbol{f}, \boldsymbol{m}, \boldsymbol{\sigma} \geq 0 \end{cases} \tag{5.5}$$

where $v(p_i, t_j)$ are slack variables. These slack variables give the control action for each transition. For more details on this topic, see ([MRRS08]).

5.2 A traffic system

Here, a hybrid PN model, that represents a traffic system consisting of two (one-way streets) intersections connected by a (one-way street) link, is introduced and simulated. The proposed example is shown in fig. 3.1 (studied in [VSBS10] for control purposes). In this, the dynamic of the vehicles is represented by continuous nodes, while the traffic lights are modeled as discrete.



(a) (b)

Figure 5.2: Queue lengths at intersection 2 obtained with delays for $\{t_5^2, t_7^2, t_9^2\}$ as a) $\{20, 20, 0.1\}$ and b) $\{14, 26, 29\}$.

Let us firstly explain the model of one intersection. Places $\{p_1^1, p_2^1\}$ represent the queues of vehicles before crossing the intersection 1. Cars arrive through $\{t_1^1, t_3^1\}$, being transitions of type ic constrained by self-loops $\{p_3^1, p_4^1\}$ that represent the number of servers (street lanes). Vehicles depart through t_2^1 or t_4^1 (type *ic*) when the traffic light enabled them, i.e., when there is a token in p_5^1 or p_7^1, respectively. The traffic light for this intersection is represented by nodes $\{p_5^1, p_6^1, p_7^1, p_8^1, t_5^1, t_6^1, t_7^1, t_8^1\}$, which describe the sequence of the traffic light stages. In this, the transitions are of type dd. A token in p_5^1 means a green signal for the queue p_1^1, but red for p_2^1.

Similarly, a token in p_7^1 represents a green signal for the queue p_2^1 but red for p_1^1. Places p_6^1 and p_8^1 represent intermediate stages when the traffic light is amber for one queue but red for the other, so no car can cross the intersection. Similarly, nodes with the superscript 2, i.e., $\{p_x^2, t_x^2\}$), represent the nodes of the second intersection and its traffic light. In this, the place p_9^2 and the transition t_9^2 (type *dd*) have been added in order to simulate the offset, i.e., the relative time between the cycles of both traffic lights, which is given by the delay of t_9^2. The output flow of intersection 1 feeds the second one through a link, which imposes a constant delay (given by the delay of t_{11}). A detailed explanation of the link model can be found in [VSBS10]. Let us just mention here that, due to the traffic light, vehicles departing intersection 1 describe a bursting signal (like platoons or batches of cars) that arrives to the second intersection through t_1^2.

Let us consider the following delays: for $\{t_5^1, t_6^1, t_7^1, t_8^1\}$ the delays are $(20, 5, 20, 4)$ seconds (in the same order). For $\{t_1^1, t_2^1, t_3^1, t_4^1\}$ are $(1, 1/3, 1/3, 1/5)$ seconds, for $\{t_1^2, t_2^2, t_3^2, t_4^2\}$ are $(1/3, 1/5, 1, 1/3)$ seconds, and for the link $\{t_9, t_{10}, t_{11}, t_{12}\}$ are $(1/10, 1/3, 30, 1/3)$ seconds (the link delay is $\theta_{11} = 30$ seconds). The initial marking is as described in fig. 3.1.

The goal in this example is to obtain, through simulations, suitable switching delays for the second traffic light, in order to reduce the queue lengths at intersection 2. The parameters to optimize are the green periods (amber periods are fixed and equal to $\theta_6^2 = 5$ and $\theta_8^2 = 4$), i.e., the delays of t_5^2, t_7^2 , and the offset represented by the delay of t_9^2. Fig. 5.2 shows the evolution of the queues for the first 400 seconds, for two cases: a) with green periods 20 seconds and no offset, and b) with green periods 14 for the queue p_1^2 and 26 for the queue p_2^2 and with an offset of 29 seconds. Note that the second combination of parameters provide shorter queues. For this, the effect of the offset is very important. This example shows that simulations based on hybrid PN models can provide information about the optimal parameters for traffic lights (duration of stages and offset), in order to improve the performance in neighboring traffic intersections.

5.3 A biochemical system

This section presents and simulates a biochemical system modeled by continuous Petri nets. In most chemical models, the different amounts of chemical substances are expressed in terms of concentrations rather than as whole numbers of molecules. This implies that the state of the system is given by a vector of real numbers, where each component represents the concentration of a compound. On the other hand, the dynamics of most reactions is driven by the mass action law, what roughly implies that the speed of a reaction is proportional to the product of the concentrations of the reactants. These facts make continuous Petri nets under product semantics an appealing modeling formalism for biochemical systems.

The net system in Fig. 5.3 models a signaling pathway described and studied in [CSK$^+$03]. More precisely, the net is a graphical representation of the Extracellular signal Regulated Kinase (ERK) signaling pathway regulated by Raf Kinase Inhibitor Protein (RKIP). The marking of each place represents the concentration of the compound associated to it, and the transitions represent the different chemical reactions that take place (see [CSK$^+$03] for a more detailed description of the pathway). Notice that, although the net has conflicts, the assumed product semantics fully determines the flows of all continuous transitions, and therefore it is not necessary to impose a conflict resolution policy.

Since the state of the system is expressed as concentration levels, every transition is considered continuous and product server semantics is adopted. The parameter $\boldsymbol{\lambda}$ estimated in [CSK$^+$03] is $\boldsymbol{\lambda} = [0.53,\ 0.0072,\ 0.625,\ 0.00245,\ 0.0315,\ 0.8,\ 0.0075,\ 0.071,\ 0.92,\ 0.00122,\ 0.87]$. As initial concentrations of the compounds we take the following values: $\boldsymbol{m}_0 = [2,\ 2.5,\ 0,\ 0,\ 0,\ 0,\ 2.5,\ 0,\ 2.5,\ 3,\ 0]$.

Figures 5.4 and 5.5 show the time evolution of some of the compounds in the system along 40 time units. In particular, Fig. 5.4 shows the dynamics of Raf-1*, RKIP and their complex Raf-1*/RKIP, and Fig. 5.5 shows the activity of

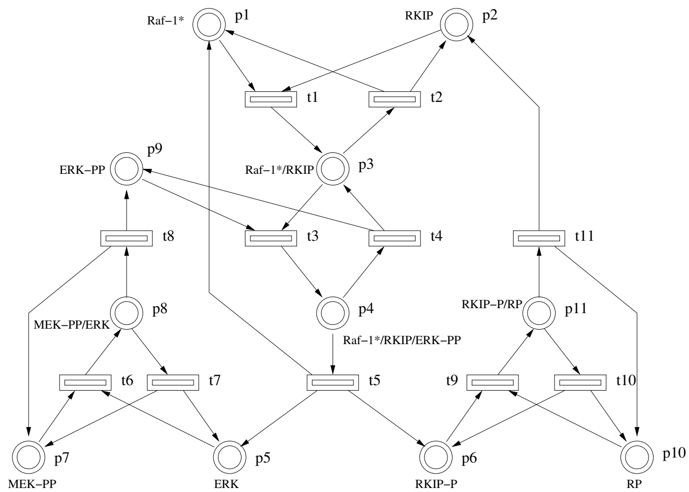


Figure 5.3: Petri net modeling the ERK signaling pathway regulated by RKIP.

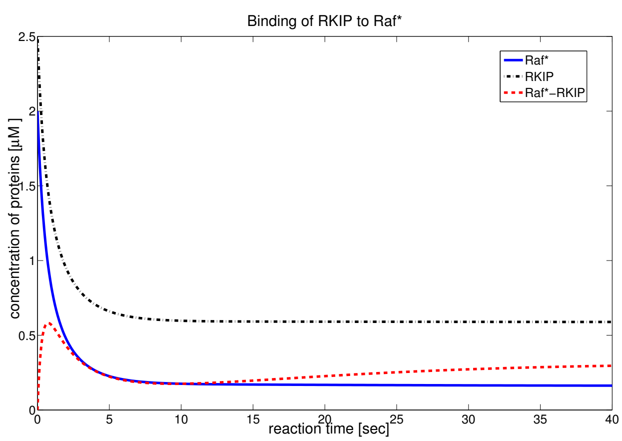


Figure 5.4: Time evolution of Raf-1*, RKIP and their complex Raf-1*/RKIP.

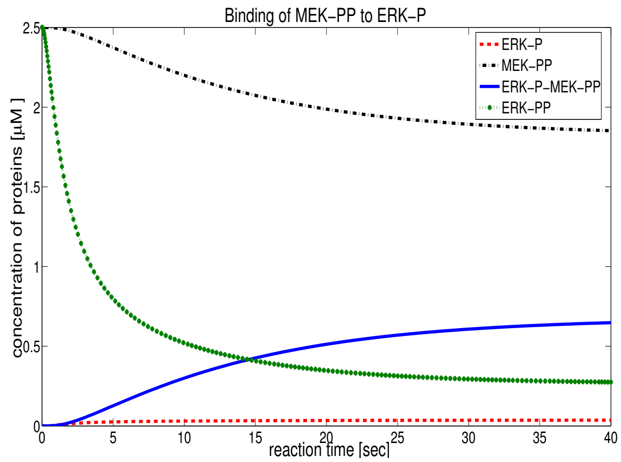


Figure 5.5: Activity of MEK-PP which phosphorylates and activates ERK.

MEK-PP which phosphorylates and activates ERK. As discussed in [CSK+03], intensive simulations can be used to perform sensitivity analysis with respect to the variation of initial conditions.

5.4 A Kanban-like flexible manufacturing system

Here, we consider the Kanban-like flexible manufacturing system (FMS) consisting of two workflows that are attended by a pool of three machines (see Fig. **??**), to perform a structural controllability analysis. A TCPN that models the system is presented in Fig. **??** (this model is available in the models folder as Model_FMS_3.mat). It has 216 configurations and 11 transitions, each one representing one of the following events:

- Loading of material to the machines (t_1, t_3, t_5, t_7, t_9).
- Unloading of the processed material to be stored in a buffer ($t_2, t_4, t_6, t_8, t_{10}$).
- Removing parts from the system output (t_{11}).

The timing of the net is given by $\lambda = \left[1\ \frac{1}{3}\ 1\ \frac{1}{4}\ 1\ \frac{1}{3}\ 1\ \frac{1}{5}\ 1\ \frac{1}{5}\ 1\right]^T$. We work based on the assumption that machines are consistently working at their nominal speed and are freed up as soon as the material is processed. As a result, we have identified $T_{nc} = \{t_2, t_4, t_6, t_8, t_{10}\}$. The remaining events are controllable because it is possible to decide when to use a specific machine to process a particular piece, and we can always empty the output buffer. This leads to T_c being $T_c = \{t_1, t_3, t_5, t_7, t_9, t_{11}\}$.

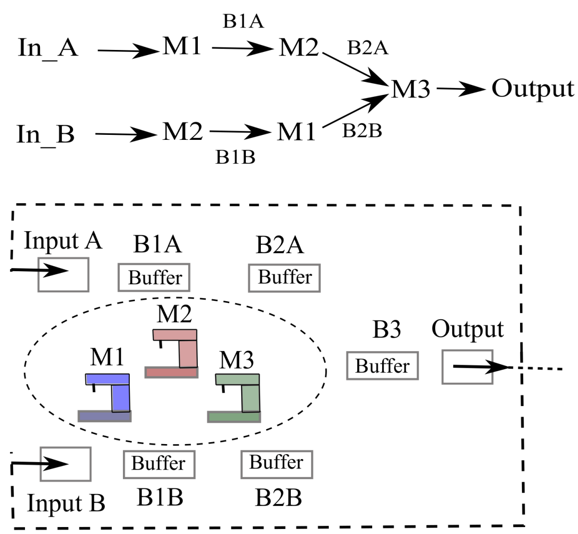


Figure 5.6: A Kanban-like flexible manufacturing system. and its production process.

Net rank-controllability analysis: Once the model has been loaded, the test can be carried out by selecting *Continuous* → *Structural controllability analysis* → *Net rank-controllability test.* We present 3 cases:
• $T_c = \{t_1, t_5\}$: We enter this set of controllable transitions on the pop-up windows as [1 5] and we obtain the message
"*Influence is not total! Therefore, the set of controllable transitions does not guarantee net rank-controllability. The only influenced nodes are: Places = [1 2 3 4 5 6 7 8 9 10 11 12 17 18] Transitions = [1 2 3 4 5 6 7 8]*"
This means that, since the property of influence does not hold, then there are configurations in which the net is not rank-controllable.
• $T_c = \{t_1, t_5, t_9, t_{11}\}$ By entering this set of controllable transitions as [1 5 9 11] we obtain the message
"*It is not possible to decide if the timed net is net rank-controllable. The condition related to the choice places is not fulfilled.*".
In this case the test cannot conclude if the system is NRC, since one of the sufficient conditions does not hold. In particular, the one related to the choice places. This serves to give indications to the operator/researcher about where the problem may be in order to guarantee controllability.
• $T_c = \{t_1, t_3, t_5, t_7, t_9, t_{11}\}$: Here we choose the set that we will consider in our case study, which is such that it satisfies all the structural conditions for controllability and is indicated by the message: "*The timed net is net rank-*

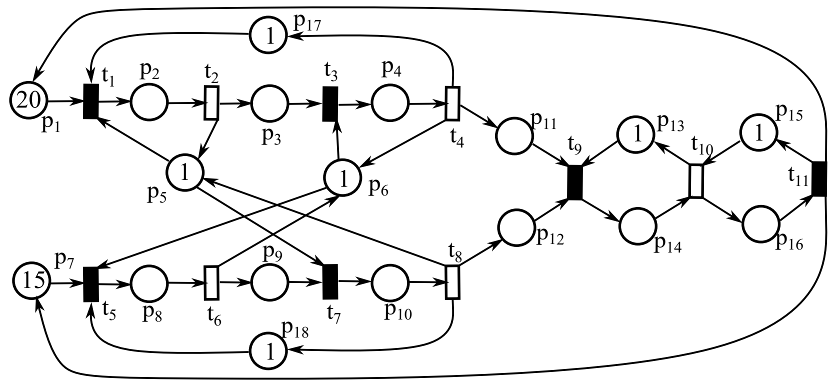


Figure 5.7: A TCPN system that models the FMS of Fig **??**. The controllable transitions are depicted as black transitions.

controllable.".

From the previous analysis, we conclude that the system is NRC. Moreover, this system is a live and bounded TCPN. Therefore, T_c also guarantees that the system will be controllable over its connected sets of equilibrium markings.

Equilibrium connectivity graph: This routine can be used to calculate the equilibrium connectivity graph of a system by selecting the menu *Continuous* $\rightarrow$ *Structural controllability analysis* $\rightarrow$ *Equilibrium connectivity graph* and entering the set of controllable transitions as the input. Each node of the resulting graph represents a configuration of the system whose region contains equilibrium markings, and the edges indicate the connection between the equilibrium markings of the different regions. This provides insights into the controllability of the system, showing the sets of equilibrium points where the controllability property holds.

Using this routine and selecting the set of controllable transitions [1 3 5 7 9 11], a matrix E_conf is obtained. Each row of this matrix represents a configuration of the system in which equilibrium markings are present in the corresponding regions. For this particular case, the results are the following:

E_conf =

1 2 3 4 7 8 9 10 13 14 16 (node 1)   1 2 3 4 7 8 9 10 13 15 16 (node 2)
1 2 3 4 18 8 9 10 12 14 16 (node 3)   1 2 3 4 18 8 9 10 12 15 16 (node 4)
1 2 3 4 18 8 9 10 13 14 16 (node 5)   1 2 3 4 18 8 9 10 13 15 16 (node 6)
17 2 3 4 7 8 9 10 11 14 16 (node 7)   17 2 3 4 7 8 9 10 11 15 16 (node 8)
17 2 3 4 7 8 9 10 13 14 16 (node 9)   17 2 3 4 7 8 9 10 13 15 16 (node 10)
17 2 3 4 18 8 9 10 11 14 16 (node 11)   17 2 3 4 18 8 9 10 11 15 16 (node 12)
17 2 3 4 18 8 9 10 12 14 16 (node 13)   17 2 3 4 18 8 9 10 12 15 16 (node 14)

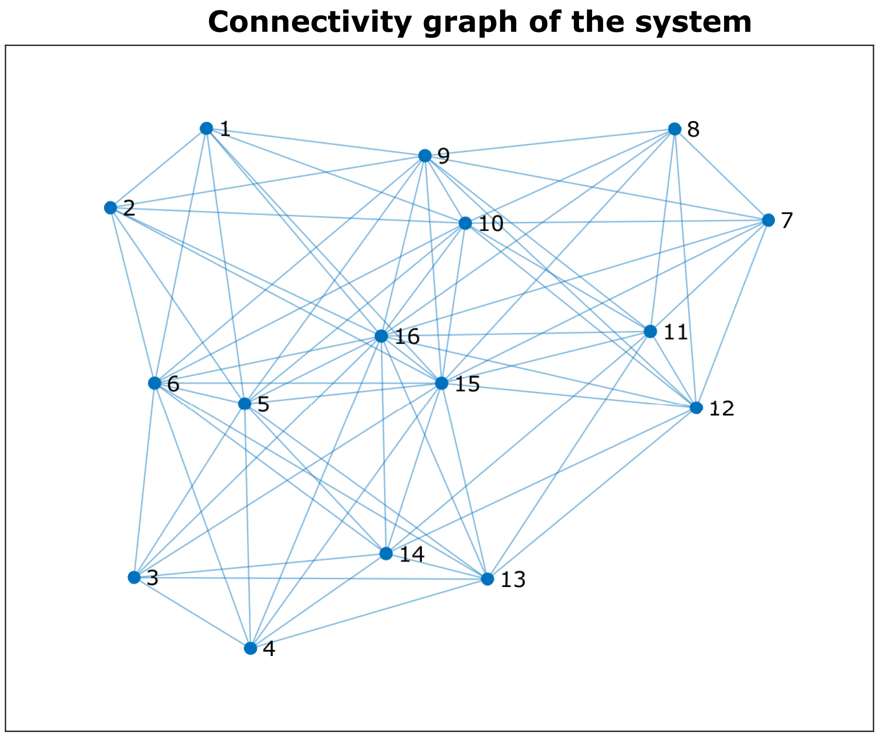


Figure 5.8: Equilibrium connectivity graph of the FMS system. Each node corresponds to a single region of the TCPN system in which it contains equilibrium markings.

17 2 3 4 18 8 9 10 13 14 16 (node 15) 17 2 3 4 18 8 9 10 13 15 16 (node 16)

For instance, the region corresponding to the configuration $\mathcal{C}_1 = \{(p_1, t_1), (p_2, t_2), (p_3, t_3), (p_4, t_4), (p_7, t_5), (p_8, t_6), (p_9, t_7), (p_{10}, t_8), (p_{13}, t_9), (p_{14}, t_{10}), (p_{16}, t_{11})\}$ (node 1) contains equilibria. Furthermore, the routine generates a variable-type graph called E_graph, which has 16 nodes and 76 edges, representing the connectivity of the equilibrium sets. This graph can be plotted (as in Fig. **??**), providing a visual representation of the equilibrium connectivity of the system. For this particular case, the equilibrium sets in all the regions are connected, meaning that the system exhibits the controllability property over all of its equilibrium markings.

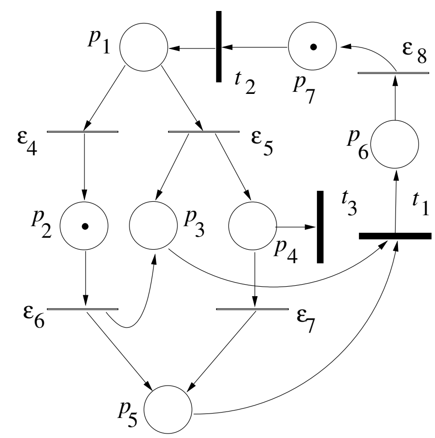


Figure 5.9: The Petri net system considered in Examples 5, 8, 10 and 12 of [MSCS12].

5.5 Fault Diagnosis with continuous Petri nets

The following examples illustrating the fault diagnosis procedure using untimed continuous Petri nets have been considered in [MSCS12].

5.5.1 Example 1

Let us consider the Petri net in Figure 5.6 with:

$$T_o = \{t_1, t_2, t_3\}, \quad T_u = \{\varepsilon_4, \varepsilon_5, \varepsilon_6, \varepsilon_7, \varepsilon_8\} \quad and \quad T_f^1 = \{\varepsilon_5\}.$$

The matrices Pre, $Post$ and m_0 can be imported in $SimHPN$ toolbox using $diagnosis1.mat$ file from the $Models$ folder. Places and transitions follow the numeration in Figure 5.6. The following input parameters should be used:

- Number of observable transitions: 3
- Sequence of observed transitions: [1 2 3]
- Sequence of observed firing quantity of transitions: [0.7 0.5 0.5]
- Number of fault classes: 1
- Transitions in fault class 1: 5

In such a case we have only one fault class containing transition ε_5. Note that, in the matrices Pre and $Post$ the observable transitions have to occupy the first columns. We are considering the following observation: $t_1(0.7)t_2(0.5)t_3(0.5)$. The following results are shown at the MATLAB command prompt:

```
Number of observable transitions: 3
First 3 transitions are observable and the rest not
Fault class Tf^{1}={\epsilon_5}

Observed sequence: t1(0.7)t2(0.5)t3(0.5)
=========================================
==========press a key to continue========
=========================================

Enabling bound of t4 is 2
Enabling bound of t5 is 2
Enabling bound of t6 is 2
Enabling bound of t7 is 2
Enabling bound of t8 is 2

***********************************************************
For empty word:
***********************************************************

Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [0 1 0 0 0 0 1 0 0 0 0 0]'
e_2 = [0 0 1 0 1 0 1 0 0 1 0 0]'

Computing the diagnosis states
Diagnosis state for Tf^{1}: N

***********************************************************
Observed sequence:  t1(0.70)
***********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [0 0.3 0.7 0 0.7 0 1 0 0 0.7 0 0]'
e_2 = [0 0 1 0 1 0 1 0 0 1 0 0]'
***********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [0 0 0.3 0 0.3 0 1.7 0 0 1 0 0.7]'
e_2 = [0 0.3 0 0 0 0 1.7 0 0 0.7 0 0.7]'
e_3 = [0 0.3 0 0 0 0.7 1 0 0 0.7 0 0]'
e_4 = [0 0 0.3 0 0.3 0.7 1 0 0 1 0 0]'
Computation time:                 0.01 seconds
Computing the diagnosis states
```

```
Diagnosis state for Tf^{1}: N

**********************************************************
Observed sequence:  t1(0.70) t2(0.50)
**********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [0 0.3 0 0 0 0 1.7 0 0 0.7 0 0.7]'
e_2 = [0 0 0.3 0 0.3 0 1.7 0 0 1 0 0.7]'
e_3 = [0 0 0.3 0 0.3 0.7 1 0 0 1 0 0]'
e_4 = [0 0.3 0 0 0 0.7 1 0 0 0.7 0 0]'
**********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [0 0.3 0.5 0 0.5 0 1.2 0 0.5 0.7 0.5 0.7]'
e_2 = [0.5 0.3 0 0 0 0 1.2 0 0 0.7 0 0.7]'
e_3 = [0 0.3 0.5 0.5 0 0 1.2 0 0.5 0.7 0 0.7]'
e_4 = [0 0.8 0 0 0 0 1.2 0.5 0 0.7 0 0.7]'
e_5 = [0 0 0.8 0 0.8 0 1.2 0.5 0 1.5 0 0.7]'
e_6 = [0 0 0.8 0.5 0.3 0 1.2 0 0.5 1 0 0.7]'
e_7 = [0.5 0 0.3 0 0.3 0 1.2 0 0 1 0 0.7]'
e_8 = [0 0 0.8 0 0.8 0 1.2 0 0.5 1 0.5 0.7]'
e_9 = [0 0 0.8 0 0.8 0.7 0.5 0.5 0 1.5 0 0]'
e_10 = [0 0 0.8 0.5 0.3 0.7 0.5 0 0.5 1 0 0]'
e_11 = [0.5 0 0.3 0 0.3 0.7 0.5 0 0 1 0 0]'
e_12 = [0 0 0.8 0 0.8 0.7 0.5 0 0.5 1 0.5 0]'
e_13 = [0 0.8 0 0 0 0.7 0.5 0.5 0 0.7 0 0]'
e_14 = [0 0.3 0.5 0.5 0 0.7 0.5 0 0.5 0.7 0 0]'
e_15 = [0.5 0.3 0 0 0 0.7 0.5 0 0 0.7 0 0]'
e_16 = [0 0.3 0.5 0 0.5 0.7 0.5 0 0.5 0.7 0.5 0]'
Computation time:                 0.01 seconds
Computing the diagnosis states
Diagnosis state for Tf^{1}: U

**********************************************************
Observed sequence:  t1(0.70) t2(0.50) t3(0.50)
**********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [0 0.3 0.5 0.5 0 0 1.2 0 0.5 0.7 0 0.7]'
e_2 = [0 0 0.8 0.5 0.3 0 1.2 0 0.5 1 0 0.7]'
e_3 = [0 0 0.8 0.5 0.3 0.7 0.5 0 0.5 1 0 0]'
```

```
e_4 = [0 0.3 0.5 0.5 0 0.7 0.5 0 0.5 0.7 0 0]'
**********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [0 0 0.8 0 0.3 0 1.2 0 0.5 1 0 0.7]'
e_2 = [0 0.3 0.5 0 0 0 1.2 0 0.5 0.7 0 0.7]'
e_3 = [0 0.3 0.5 0 0 0.7 0.5 0 0.5 0.7 0 0]'
e_4 = [0 0 0.8 0 0.3 0.7 0.5 0 0.5 1 0 0]'
Computation time:                 0.00 seconds
Computing the diagnosis states
Diagnosis state for Tf^{1}: F
```

5.5.2 Example 2: (a manufacturing system)

Let us consider the Petri net in Fig. 5.7 where transitions t_1 to t_{12} correspond to observable events, transitions ε_{13} to ε_{24} correspond to unobservable but regular events while ε_{25} and ε_{26} are unobservable and faulty transitions. In particular we assume $T_f^1 = \{\varepsilon_{25}\}$ and $T_f^2 = \{\varepsilon_{26}\}$.

The matrices Pre, $Post$ and m_0 are defined in the MATLAB file *Diagnosis_2.mat* and can be imported in *SimHPN* toolbox using the menu *File* $\rightarrow$ *Import from .mat file*. Places and transitions follow the numeration in Fig. 5.7. The following parameters will be considered:

- Number of observable transitions: 12
- Sequence of observed transitions: [1 1 2 12 3 12 3 6 7 8 4 5 9 10 11]
- Sequence of observed firing quantity of transitions: [1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
- Number of fault classes: 1
- Transitions in fault class 1: [25 26]

This net system has two fault classes. The first one contains transition ε_{25} and the second one contains transition ε_{26}. The cardinality of the set of observable transitions is 12 and the considered observation is $w = t_1(1)t_1(1)t_2(1)t_{12}(1)t_3(1)t_{12}(1)t_3(1)t_6(1)t_7(1)t_8(1)t_4(1)t_5(1)t_9(1)t_{10}(1)t_{11}(1)$. The fault state after each observation is shown at the MATLAB command prompt.

5.5.3 Example 3: (unobservable acyclic subnet)

Let us consider the Petri net in Figure 5.8 where transitions t_1 to t_6 correspond to observable events, transitions ε_7 and ε_8 correspond to unobservable but regu-

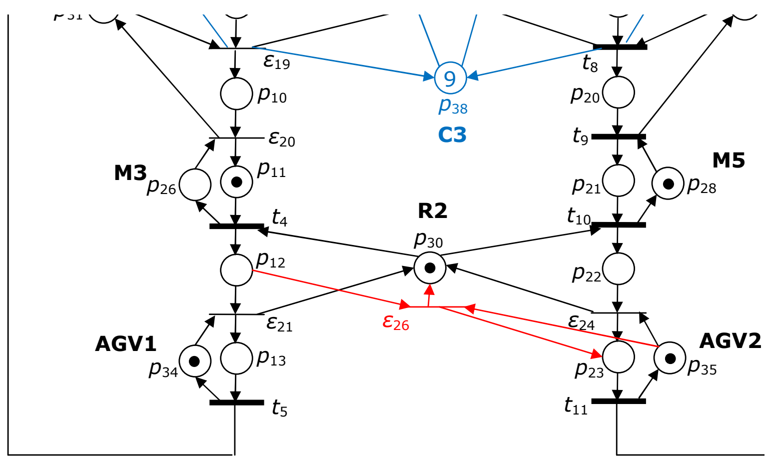


Figure 5.10: Petri net model of the manufacturing system considered in Subsection VII.A.

Figure 5.11: The Petri net considered in Subsection VII.B.

lar events while ε_9 and ε_{10} are unobservable and faulty transitions. In particular we assume $T_f^1 = \{\varepsilon_9\}$ and $T_f^2 = \{\varepsilon_{10}\}$.

The matrices *Pre*, *Post* and m_0 can be imported from *Diagnosis_3.mat* file. Places and transitions follow the numeration in Fig. 5.7 and the following parameters are used:

- Number of observable transitions: 6
- Sequence of observed transitions: [1 2 5 4 1 3]
- Sequence of observed firing quantity of transitions: [0.1 0.1 0.3 0.1 0.1 0.1]
- Number of fault classes: 2
- Transitions in fault class 1: [9]
- Transitions in fault class 2: [10]

Such net has two fault classes. The first one contains transition ε_9 and the second one contains transition ε_{10}. The cardinality of the set of observable transitions is 6 and the considered observation is $w = t_1(0.1)t_2(0.1)t_5(0.3)t_4(0.1)t_1(0.1)t_3(0.1)$. The following results are obtained at the MATLAB prompt:

```
Number of observable transitions: 6
First 6 transitions are observable and the rest not
```

```
Fault class Tf^{1}={\epsilon_9}
Fault class Tf^{2}={\epsilon_10}

Observed sequence: t1(0.1)t2(0.1)t5(0.3)t4(0.1)t1(0.1)t3(0.1)
=========================================
==========press a key to continue========
=========================================

Enabling bound of t7 is 2.000000e+00
Enabling bound of t8 is 2.000000e+00
Enabling bound of t9 is 2
Enabling bound of t10 is 2.000000e+00

***********************************************************
For empty word:
***********************************************************

Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [2 2 0 0 0 0 0 0 0 0 0 0 0]'

Computing the diagnosis states
Diagnosis state for Tf^{1}: N
Diagnosis state for Tf^{2}: N

***********************************************************
Observed sequence:  t1(0.10)
***********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [2 2 0 0 0 0 0 0 0 0 0 0 0]'
***********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [1.6 2 0 0 0 0 0.2 0 0.1 0 0.1 0]'
e_2 = [1.6 2 0.1 0 0 0 0.1 0 0.1 0 0 0]'
e_3 = [1.6 2 0.1 0.1 0 0 0 0 0 0 0 0]'
e_4 = [1.6 2 0 0.1 0 0 0.1 0 0 0 0.1 0]'
Computation time:                 0.01 seconds
Computing the diagnosis states
Diagnosis state for Tf^{1}: U
Diagnosis state for Tf^{2}: N
```

```
*********************************************************
Observed sequence:  t1(0.10) t2(0.10)
*********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [1.6 2 0.1 0 0 0 0.1 0 0.1 0 0 0]'
e_2 = [1.6 2 0 0 0 0 0.2 0 0.1 0 0.1 0]'
e_3 = [1.6 2 0 0.1 0 0 0.1 0 0 0 0.1 0]'
e_4 = [1.6 2 0.1 0.1 0 0 0 0 0 0 0 0]'
*********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [1.6 1.6 0 0.1 0 0.1 0.2 0 0 0.1 0.1 0]'
e_2 = [1.6 1.6 0.1 0.1 0 0.1 0.1 0 0 0.1 0 0]'
e_3 = [1.6 1.6 0.1 0 0 0.1 0.2 0 0.1 0.1 0 0]'
e_4 = [1.6 1.6 0 0 0 0.1 0.3 0 0.1 0.1 0.1 0]'
e_5 = [1.6 1.6 0.1 0 0.1 0.1 0.1 0 0.1 0 0 0]'
e_6 = [1.6 1.6 0 0 0.1 0.1 0.2 0 0.1 0 0.1 0]'
e_7 = [1.6 1.6 0.1 0 0 0.2 0.1 0 0.1 0 0 0.1]'
e_8 = [1.6 1.6 0 0 0 0.2 0.2 0 0.1 0 0.1 0.1]'
e_9 = [1.6 1.6 0.1 0.1 0.1 0.1 0 0 0 0 0 0]'
e_10 = [1.6 1.6 0 0.1 0.1 0.1 0.1 0 0 0 0.1 0]'
e_11 = [1.6 1.6 0.1 0.1 0 0.2 0 0 0 0 0 0.1]'
e_12 = [1.6 1.6 0 0.1 0 0.2 0.1 0 0 0 0.1 0.1]'
Computation time:                  0.01 seconds
Computing the diagnosis states
Diagnosis state for Tf^{1}: U
Diagnosis state for Tf^{2}: U

*********************************************************
Observed sequence:  t1(0.10) t2(0.10) t5(0.30)
*********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [1.6 1.6 0 0 0 0.1 0.3 0 0.1 0.1 0.1 0]'
*********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [1.9 1.9 0 0 0 0.1 0 0 0.1 0.1 0.1 0]'
Computation time:                  0.00 seconds
Computing the diagnosis states
Diagnosis state for Tf^{1}: F
Diagnosis state for Tf^{2}: N
```

```
**********************************************************
Observed sequence:  t1(0.10) t2(0.10) t5(0.30) t4(0.10)
**********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [1.9 1.9 0 0 0 0.1 0 0 0.1 0.1 0.1 0]'
**********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [1.9 1.9 0 0 0 0 0 0.1 0.1 0.1 0.1 0]'
Computation time:                0.00 seconds
Computing the diagnosis states
Diagnosis state for Tf^{1}: F
Diagnosis state for Tf^{2}: N

**********************************************************
Observed sequence:  t1(0.10) t2(0.10) t5(0.30) t4(0.10) t1(0.10)
**********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [1.9 1.9 0 0 0 0 0 0.1 0.1 0.1 0.1 0]'
**********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
e_1 = [1.5 1.9 0 0 0 0 0.2 0.1 0.2 0.1 0.2 0]'
e_2 = [1.5 1.9 0.1 0 0 0 0.1 0.1 0.2 0.1 0.1 0]'
e_3 = [1.5 1.9 0.1 0.1 0 0 0 0.1 0.1 0.1 0.1 0]'
e_4 = [1.5 1.9 0 0.1 0 0 0.1 0.1 0.1 0.1 0.2 0]'
Computation time:                0.00 seconds
Computing the diagnosis states
Diagnosis state for Tf^{1}: F
Diagnosis state for Tf^{2}: N

**********************************************************
Observed sequence:  t1(0.10) t2(0.10) t5(0.30) t4(0.10) t1(0.10) t3(0.10)
**********************************************************

Vertices of the set of consistent markings after cutting
e_1 = [1.5 1.9 0.1 0 0 0 0.1 0.1 0.2 0.1 0.1 0]'
e_2 = [1.5 1.9 0.1 0.1 0 0 0 0.1 0.1 0.1 0.1 0]'
**********************************************************
Vertices of the set of consistent markings (vertices of \bar Y(m_0,w))
```

```
e_1 = [1.5 1.9 0 0 0 0 0.1 0.2 0.2 0.1 0.1 0]'
e_2 = [1.5 1.9 0 0.1 0 0 0 0.2 0.1 0.1 0.1 0]'
Computation time:                 0.00 seconds
Computing the diagnosis states
Diagnosis state for Tf^{1}: F
Diagnosis state for Tf^{2}: N
```

Bibliography

[AD98] H. Alla and R. David. Continuous and hybrid Petri nets. *Journal of Circuits, Systems, and Computers*, 8(1):159–188, 1998.

[BMG00] F. Balduzzi, G. Menga, and A. Giua. First-order hybrid Petri nets: a model for optimization and control. *IEEE Trans. on Robotics and Automation*, 16(4):382–399, 2000.

[CSK$^+$03] K.-H. Cho, S.-Y. Shin, H.-W. Kim, O. Wolkenhauer, B. McFerran, and W. Kolch. Mathematical modeling of the influence of rkip on the erk signaling pathway. In Corrado Priami, editor, *Computational Methods in Systems Biology*, volume 2602 of *Lecture Notes in Computer Science*, pages 127–141. Springer Berlin, Heidelberg, 2003.

[DA10] R. David and H. Alla. *Discrete, Continuous and Hybrid Petri Nets.* Springer-Verlag, 2010. 2^{nd} edition.

[DHP$^+$93] F. DiCesare, G. Harhalakis, J. M. Proth, M. Silva, and F. B. Vernadat. *Practice of Petri Nets in Manufacturing.* Chapman & Hall, 1993.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Interactability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, 1979.

[HGD08] M. Heiner, D. Gilbert, and R. Donaldson. Petri nets for systems and synthetic biology. In M. Bernardo, P. Degano, and G. Zavattaro, editors, *Formal Methods for Computational Systems Biology*, Lecture Notes in Computer Science, pages 215–264. Springer Berlin, Heidelberg, 2008.

[JM10] J. Júlvez and C. Mahulea. SimHPN: a MATLAB toolbox for continuous Petri nets. In *Proc. of the* 10^{th} *Workshop on Discrete Event Systems*, pages 24–29, Berlin, Germany, August 2010.

[JMV11] J. Júlvez, C. Mahulea, and C.R. Vázquez. Analysis and Simulation of Manufacturing Systems using SimHPN toolbox. In *Proc. of the 7th IEEE Conf. on Automation Science and Engineering*, pages 432–437, Trieste, Italy, August 2011.

[JMV12] Jorge Júlvez, Cristian Mahulea, and Carlos-Renato Vázquez. Simhpn: A matlab toolbox for simulation, analysis and design with hybrid petri nets. *Nonlinear Analysis: Hybrid Systems*, 6(2):806 – 817, 2012.

[JRS05] J. Júlvez, L. Recalde, and M. Silva. Steady-state performance evaluation of continuous mono-T-semiflow Petri nets. *Automatica*, 41(4):605–616, 2005.

[MMP03] M. Matcovschi, C. Mahulea, and O. Pastravanu. Petri Net Toolbox for MATLAB. In 11^{th} *IEEE Mediterranean Conference on Control and Automation MED'03*, Rhodes, Greece, July 2003.

[MRRS08] C. Mahulea, A. Ramírez, L. Recalde, and M. Silva. Steady state control reference and token conservation laws in continuous Petri net systems. *IEEE Trans. on Autom. Science and Engineering*, 5(2):307–320, 2008.

[MRS09] C. Mahulea, L. Recalde, and M. Silva. Basic Server Semantics and Performance Monotonicity of Continuous Petri Nets. *Discrete Event Dynamic Systems: Theory and Applications*, 19(2):189 – 212, June 2009.

[MSCS12] C. Mahulea, C. Seatzu, M.P. Cabasino, and M. Silva. Fault Diagnosis of Discrete-Event Systems using Continuous Petri Nets. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 2012. in press.

[Mur89] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[SGS08] F. Sessego, A. Giua, and C. Seatzu. HYPENS: a Matlab tool for timed discrete, continuous and hybrid Petri nets. In *Application and Theory of Petri Nets 2008*, volume 5062 of *Lecture Notes in Computer Science*, pages 419–428. Springer-Verlag, 2008.

[Sil85] M. Silva. *Las Redes de Petri: en la Automática y la Informática.* AC, 1985.

[SR00] M. Silva and L. Recalde. Réseaux de Petri et relaxations de l'integralité: Une vision des réseaux continus. In *Conférence Internationale Francophone d'Automatique (CIFA 2000)*, pages 37–48, 2000.

[SR02] M. Silva and L. Recalde. Petri nets and integrality relaxations: A view of continuous Petri nets. *IEEE Trans. on Systems, Man, and Cybernetics*, 32(4):314–327, 2002.

[VSBS10] C.R. Vázquez, H.Y. Sutarto, R. Boel, and M. Silva. Hybrid petri net model of a traffic intersection in an urban network. In *2010 IEEE Multiconference on Systems and Control*, Yokohama, Japan, 09/2010 2010.

[ZK95] A. Zimmermann and M. Knoke. TimeNETsim - a parallel simulator for stochastic Petri nets. In *Proc. 28th Annual Simulation Symposium*, pages 250–258, Phoenix, AZ, USA, 1995.

[ZRS01] A. Zimmermann, D. Rodríguez, and M. Silva. A Two Phase Optimisation Method for Petri Net Models of Manufacturing Systems. *Journal of Intelligent Manufacturing*, 12(5):421–432, October 2001.