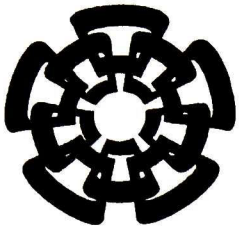
CT-906-551
Doc. 2016



Centro de Investigación y de Estudios Avanzados

del Instituto Politécnico Nacional.

Unidad Guadalajara

Estabilidad de Sistemas de Eventos Discretos Concurrentes modelados con redes de Petri.

Tesis que presenta:
Alberto Lutz Ley

Para obtener el grado de:
Doctor en Ciencias

En la especialidad de:
Ingeniería Eléctrica

Director de Tesis:
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco,Julio 2015.

Estabilidad de Sistemas de Eventos Discretos Concurrentes modelados con redes de Petri.

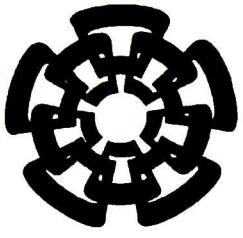
Tesis de Doctor en Ciencias
En Ingeniería Eléctrica

Por:
Alberto Lutz Ley
Maestro en Ciencias Computacionales
CINVESTAV Unidad Guadalajara 2009-2011

Beca otorgada por CONACYT, No. 234687

Asesor de Tesis:
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Julio, 2015.



Centro de Investigación y de Estudios Avanzados

del Instituto Politécnico Nacional.

Unidad Guadalajara

Stability of Concurrent Discrete Event Systems modeled as Petri nets.

A thesis presented by:
Alberto Lutz Ley

To obtain the degree of:
Doctor of Science

In the subject of:
Electrical Engineering

Thesis Advisor:
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, July 2015.

Stability of Concurrent Discrete Event Systems modeled as Petri nets.

Doctor of Science Thesis
In Electrical Engineering

By:
Alberto Lutz Ley
Master in Computer Science
CINVESTAV Unidad Guadalajara 2009-2011

Scholarship granted by CONACYT, No. 234687

Asesor de Tesis:
Dr. Luis Ernesto López Mellado

CINVESTAV del IPN Unidad Guadalajara, July, 2015.

Estabilidad de Sistemas de Eventos Discretos Concurrentes modelados con redes de Petri

Resumen

Esta tesis trata acerca del problema de estabilidad en sistemas de eventos discretos concurrentes modelados con redes de Petri (RP). En particular, el trabajo se enfoca en determinar si un sistema modelado con RP 1-acotadas tiene la propiedad de estabilidad; esta propiedad se refiere a que está garantizado que el sistema regresa, en un número finito de pasos, a un conjunto de estados que ha sido determinado como el comportamiento deseado del sistema.

Primero, la literatura relevante al problema es revisada incluyendo la definición del problema de estabilidad. Un método eficiente basado en la estructura de la red para determinar la estabilidad de una subclase de redes de Petri 1-acotadas es presentado.

Además, un procedimiento para determinar la propiedad en la clase de redes de Petri 1-acotadas es propuesto; esta aproximación está basada en el análisis del unfolding de la red y el algoritmo tiene una complejidad en tiempo polinomial con respecto al número de nodos en el unfolding.

Éste método ha sido implementado como una herramienta de software y ha sido comparado con un algoritmo basado en el grafo de alcanzabilidad de la red. La aplicación de ambos métodos es ilustrada mediante un caso de estudio.

Stability of Petri-net modeled Concurrent Discrete Event Systems

Abstract

This thesis deals with the stability problem on discrete event systems modelled as Petri nets. Particularly, this work focuses on determining if a system that has been modelled as a 1-bounded Petri net has the property of stability; this property indicates that it is guaranteed that the system will return to a subset of states that has been determined to be the desired behaviour of the system. This subset is to be reached within a finite number of event occurrences.

First, the relevant literature is reviewed and then the problem of state stability is presented. A polynomial time method based on the structure of the net for the decision of stability on a subclass of 1-bounded Petri nets is introduced.

Also, a procedure for the decision of the problem on the class of 1-bounded nets is proposed; this approach is based on the analysis of the net unfolding and the introduced algorithm has a polynomial time complexity with respect to the size of the unfolding.

The proposed unfolding-based method has been implemented as a software tool and it has been compared with an algorithm that decides the property by analysing the reachability graph. The execution of both methods is illustrated through a study case.

Acknowledgement

This research work was developed at CINVESTAV IPN, Guadalajara Unit and was possible thanks to the support of CONACYT (National Science and Technology Council of Mexico). I want to thank both institutions because they gave me the opportunity to do research and find interesting results.

I am grateful with my advisor, Dr. Luis Ernesto López Mellado, for his aid, support and guidance during the development of this research work. Also, I wish to thank Dr. Francesco Basile for allowing me to do a research stay at the University of Salerno, and for giving me useful comments regarding my work.

I really appreciate the enormous work that my parents Arturo and Guadalupe did so I could continue studying. I thank my siblings, Arturo and America for their support. I also thank my girlfriend, Ana Paula, for her continuous support and encouraging words. I am also grateful for the help that my aunt Rosa Maria kindly offered during my studies.

Index

Introduction

On very recent years, the size and complexity of man-made systems, namely computer networks and automated manufacturing processes have been increased at a very fast pace. Also, the importance of a robust implementation of those systems has become crucial for many aspects of our current lives. As technology advances, we are being increasingly surrounded by autonomous systems whose correct operation has a great influence on fundamental aspects, such as our own safety, or the huge losses related to manufacturing downtimes.

In particular, the issue of developing systems that behave predictably and correctly, under the occurrence of undesired and unavoidable events is of high importance. However, the size and complexity of current information systems has made this objective difficult and costly to reach, both in time and resources.

There have been several proposals that address these issues from computer science, control and particularly, discrete event systems (DES) communities. Among them, the concept of stability of DES, modeled as finite automata, stands out; it guarantees that once a system deviates from some ideal behavior, it returns to a correct functioning after a finite number of steps.

Stability is to be understood as a property related to the robustness or resilience of a system; a stable system can operate in the presence of abnormalities and return to a normal or desired set of states in a finite number of steps. In this way, stability guarantees that all undesired behaviour is transient; if errors or exceptions are not frequent, a stable system can cope with these occurrences while exhibiting a desired behaviour most of the time. The stability of DESs has been already studied and addressed under different approaches. Most of the published works deal with stability as a property analogous to that studied on continuous systems; The concept of stability in the sense of Lyapunov to DESs theory was introduced in [Passino, 1994][Passino, 1998]. More recently, a technique based on Lyapunov stabilization to design Petri net controllers that keep the net bounded was presented in [Retchkiman, 1999],[Retchkiman 2000] and [Retchkiman, 2011].

In contrast to the works of Lyapunov stability for DESs, our research focuses in the analysis of a behavioural property of discrete event systems (DESs) in presence of unavoidable, undesired events. Specifically, the issue of determining if a system is guaranteed to return to normal behaviour in a finite number of steps after an undesired event has occurred is the aim of this work.

This property has been studied first in [Dijkstra, 1973], who introduced the concept of self-stabilization. This work motivated research in computer sciences [Arora 1993], [Schneider, 1993] and automatic control communities [Brave, 1990] , [Özveren, 1991].

Other works addressing fault tolerance are based on notions closely related to those introduced in [Özveren, 1991]. First, the synthesis of stabilizing controllers for failure recovery was presented in [Khatab, 2002], and more recently, fault tolerant supervisory controllers were studied in [Wen, 2007][Wen, 2008][Wen, 2009]. In these proposals DESs are modelled with FA.

While finite automata have been widely used as modelling and control formalism for DES, Petri nets have an undeniable advantage in terms of representation size, especially when modelling complex behaviour as concurrency, synchronization, resource allocation, etc.

At first glance, it may seem that deciding if a Petri net is stable with respect to a desired marking subset is equivalent to the problem of determining if a given subset of markings is a *Home Space* [Esparza, 1994], however these properties are different: if a subset of markings is a Home Space, the reachability set of any marking in the system has to include at least one marking in the Home Space (it is always *possible* to reach the Home Space), while if a system is stable with respect to a subset of markings, this subset is *required* to be reached in a finite number of steps.

With the aim of taking advantage of Petri nets (PN) for specifying concurrent DESs, stability is studied. In a previous work, recoverability has been studied on interpreted PN (IPN) models; in [Lutz-Ley, 2012a] recoverability is defined and characterized as a property based on the stability concept including a useful distinction of uncontrollable events. The derived algorithms are based on the analysis of the reachability graph of the IPN model. However, the state space of a given Petri net could grow exponentially with respect to the structure of the net; this is known as the state space explosion problem. Also, the stability analysis presented in literature is based in the state space of a DES, so an important issue arises: analyzing the stability of a Petri-net modeled DES with traditional methods requires the study of the reachability graph of the net.

In this thesis the stability of concurrent DES modeled by 1-bounded Petri-net is analyzed from the structure of the model instead from the reachability graph. Two methods have been proposed.

First, the stability of a subclass of 1-bounded Petri nets is analyzed: in [Lutz-Ley, 2012b] the stability of the CLSM class of Petri nets is tested by making use of the net structure only; the resulting procedure is polynomial in time with respect to the size of the net. Later, the stability of the class of 1-bounded Petri nets is studied in [Lutz-Ley 2013b]; this proposal analyzes the unfolding of the net to determine if the system is stable. Then, the unfolding-based method is compared with an intuitive approach: obtaining the Reachability Graph of the net and testing if it is stable using a well-known method in the literature for state machines. This comparison is done for an easily-scalable Petri-net model example. The proposed algorithm is more efficient than the intuitive approach.

The remainder of this document is organized as follows. Chapter 1 gives an overview of relevant literature about stability on DES, as well as interesting works about fault tolerance. In Chapter 2, the technical background necessary to understand the rest of the document is recalled. In Chapter 3, a first method for deciding the stability property on a CLSM nets is presented. In Chapter 4 the proposal of stability analysis for 1-bounded Petri nets is introduced. Chapter 6 overviews software implementation and describes the application to a case study. Finally, in Chapter 6 current research and perspectives are pointed out.

Chapter 1
STABILITY OF DISCRETE EVENT SYSTEMS

In this chapter, different notions of stability for DES are reviewed; some of them are adapted from the continuous systems theory, while others have been originated from DES concepts. Also, relevant fault tolerance works that are related to stability are revised.

1.1. DISCRETE EVENT SYSTEMS

Discrete event systems (DES) are those systems whose behavior can be described by a finite or infinite set of possible states that are numerable and distinct between them. Also, they have a set of events that make the system evolve from one state to another; the events occur in an asynchronous fashion.

The objective of DES supervisory control is to enforce some specification on a system while trying to keep its behavior as unrestricted as possible. However, this is not always possible, because there are some events like faults, errors, and exceptions which can disturb the behavior of the system; this provokes that the DES reaches some state or states that are not expected to be reached in the normal operation. In this context there are two interesting problems: a) to know whether or not the system is going to eventually reach a state of no failure and b) to know if it is possible to lead the system to a state of no failure by means of control. These two problems are intrinsically related to the concept of stability.

1.2. STABILITY

There have been some different approaches for stability of DES. Some authors have adapted concepts from continuous systems theory, while others have defined stability concepts based on DES notions only.

1.2.1. LYAPUNOV STABILITY ON DISCRETE EVENT SYSTEMS

In his work, K. Passino presents in [Passino, 1994], [Passino, 1998] the concept of Lyapunov stability adapted to DES from continuous systems. According to this approach, a discrete event system is the 5-tuple $G = (X, E, f_e, g, E_v)$ where

X is the set of states of the system
E is the set of events
$f_e: X \rightarrow X$, is the state transition function
$g: X \rightarrow P(E) - \{\emptyset\}$ is the enabling function, which indicates the events enabled in some state
E_v denotes the possible event sequences
$E_a \subset E_v$ denotes the permitted event sequences

Also, $\rho(x, y)$ represents the distance between states x, y, and $S(Xz; r)$ represents the set of states which have a distance less or equal than r to the set of states Xz.

The function $X(x_0, E_k, k)$ is used to denote the state reached after the event sequence E_k in step k starting from x_0. For some x_0, functions $X(x_0, E_k, k)$ are called *motions*.

An invariant state set X_m is stable in the sense of Lyapunov with respect to E_a , if for every $\varepsilon > 0$ it is possible to find $\delta > 0$ such that:

If $\rho(x_0, X_m) < \delta$ then $\rho(X(x_0, E_k, k), X_m) < \varepsilon$ holds
For all E_k such that $E_k E \subset E_a(x_0)$ y $k \geq 0$.

In other words, according to Passino, an invariant set of states is stable in the sense of Lyapunov if it is possible to find a neighborhood of radius δ such that starting from any state in that neighborhood, the system will end in a state with a distance shorter than ε, for any $\varepsilon > 0$. This is shown in figure 1.1.

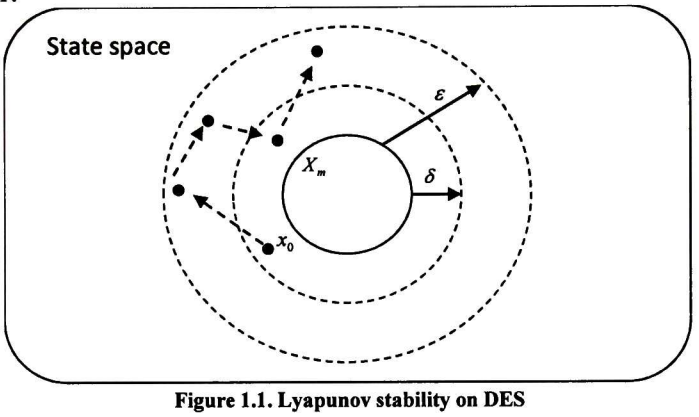


Figure 1.1. Lyapunov stability on DES

If $\rho(X(x_0, E_k, k), X_m) \to 0$ when $\to \infty$. the system is said to be asymptotically stable. Sufficient conditions for these properties are presented by K.Passino. The author also introduces the concept of Lagrange Stability for DES, which refers to the boundedness of the maximum distance achieved by motions.

In short, Passino presents the concepts of Lyapunov stability and asymptotic stability that are adapted to DES from continuous systems theory. These concepts express the notion of staying within the vicinity of a defined state set, and while being related to the idea of going back to a desired state, it does not exactly capture the concept of stability that is of interest for this work (convergence to a desired state rather than staying near the state).

1.2.2. STABILIZATION USING LYAPUNOV METHODS

Retchkiman presents the problem of stabilization of Petri nets in the sense of Lyapunov in [Retchkiman, 1999], [Retchkiman, 2000]. In this proposal, the problem is stated as the synthesis of a control law for manipulating the firing sequences to keep the system bounded.

A Petri net is said to be stabilizable if there exists a firing sequence with a vector of firing number per transition u such that the next equality is true:

$$\Delta v = A^T u \leq 0$$

where A represents the incidence matrix of the Petri net.

Intuitively, for the net to be bounded, the effect of the firing sequence u on the marking reached should be non-incrementing. This happens when the equality presented before holds, because the state equation for Petri nets is given by:

$$M' = M + A^T u$$

Then, if the expression $A^T u$ is equal to or lower than zero, the effect on the net is that the number of tokens does not increase.

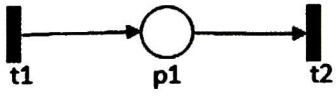


Figure 1.2. Stabilization by firing vector example

The net of figure 1.2 is an example of an unbounded net because the transition t1 can occur infinitely often, so place p1 could have an infinite number of tokens. However, if the firing of transition t1 is restricted with the firing count vector $u = [k, k]$ with $k > 0$, this is, t1 occurs the same number of times as t2, the system keeps bounded.

In a more recent work, Retchkiman approaches the same stabilization problem using the concepts of the Max-plus algebra [Retchkiman 2011]. The author conceives stabilization as a way of making a system bounded, following the concepts presented before by Passino. While keeping the system bounded can be seen as keeping the system in a set of states, this property does not capture the notion of stability that is preferred for this research, because in many cases the desired system behavior cannot be defined using boundedness only.

1.2.3. INPUT-OUTPUT STABILITY

Similarly to Passino, Tarraf adapted concepts from continuous systems to DES theory, but the concept he approached is different from Lyapunov notions [Tarraf, 2006]. The author defines a finite state machine as the 3-tuple $\{U, Y, Q\}$, and a set of mappings $(f: QxU \rightarrow Q, h: Q \rightarrow Y)$ where:

Q is the set of states
U is the finite alphabet of input symbols
Y is the alphabet of possible output values
f is the state transition function
h is the output function

It is assumed that the input and output alphabets are subsets of integer numbers.

Then, a finite state machine is said to be finite-gain input/output stable if there exist non-negative constants C, γ such that the inequality:

$$\sum_{t=0}^{T} |Y(t)| \leq C + \gamma \sum_{t=0}^{T} |U(t)|$$

holds for every time $t \geq 0$. This can be understood as follows: A system is finite gain input/output stable when the output of such system is a linear function of the input at all times. The output is then proportional to the input.

A finite state machine S is externally stable if:

S is finite-gain input/output stable

There exists a finite time $\tau > 0$ such that for every input signal $\{u(t)\}_{t=0}^{\infty}$ (infinite sequence of input symbols) and two arbitrary initial conditions of the system $q_1(0), q_2(0)$ the corresponding outputs$\{y_1(t)\}_{t=0}^{\infty}$ y $\{y_2(t)\}_{t=0}^{\infty}$ satisfy $y_1 = y_2$ for all $t \geq \tau$

This is, starting from two different initial conditions the system eventually produces the same output for a given input.

Also, conditions for a state machine to hold the properties reviewed here are presented in [Tarraf 2005] and in [Tarraf 2006]; the author continues studying these notions, and results regarding the interconnection of stable systems are presented. In addition, practical examples of the properties are shown. These notions are more related to the output response of a system and are not adequate to our purposes.

1.2.4.STABILITY ON COMPUTER SCIENCE

In 1973 Dijkstra introduced to the computer science community the notion of self-stabilization as a property that could be present on distributed systems. He defined a system as self-stabilizing when, regardless of its initial state, it is guaranteed to arrive at a legitimate state in a finite number of steps. A system which is not self-stabilizing may stay in an illegitimate state forever.

This notion of self-stabilization encompasses a formal and unified yet general approach to fault-tolerance for distributed systems. Dijkstra's work motivated research on computer science and discrete event systems regarding stability. Many of these proposals are reviewed here.

1.2.5.STATE STABILITY ON FINITE AUTOMATA

Based on Dijkstra's ideas, in [Özveren, 1991] the authors define state stability as a property of finite automata. A finite state non-deterministic automaton is defined as the tuple:

$$G = (X, \Sigma)$$

Where X is the finite set of states, and Σ is the finite set of posible events. The dynamic behavior of the system is given by functions f and d:

$$x[k+1] \in f(x[k], \sigma[k+1])$$

$$\sigma[k+1] \in d(x[k])$$

In this notation, $x[k] \in X$ is the state reached after the k-th event and $\sigma[k+1] \in \Sigma$ is the (k+1)-th event. The d function specifies what events can occur in each state. The f function tells us which is the next state and returns a set of states, being a non-deterministic automaton.

According to [Özveren, 1991] the ideal behavior is assumed as a set of event sequences that are preferred to be seen in the system behavior. For example, for a manufacturing system, such sequences could consist in the concatenation of subsequences so that the occurrence of each corresponds to the production of an individual component and a return to an idle state.

However, because of the possibility of unavoidable events like faults or errors, the actual behavior of the system could deviate from this ideal and then a successful recovery is a necessity. In order to capture this notion, a subset $E \subset X$ needs to be identified, such that returning to E means being in a state where normal or desired behavior is followed again from that point on. The stability notion then corresponds to a return to E in a finite number of steps after the system deviates from E.

This is captured in two stages. A state x is said to be pre-stable if every path starting from $x \in X$ arrives at a state in E in a finite number of transitions. A state $x \in X$ is then stable if every state in its reachability set is pre-stable.

A set of states is pre-stable with respect to E if every element on the set is pre-stable. A system is pre-stable if every primary cycle of X includes at least one state in E.

The algorithm that determines the biggest set of pre-stable states with respect to a given E is presented next:

Algorithm 1.1: findPrestableSet

Input: Automata G
Output: X_k (Maximal set of pre-stable states of G).

Let $X_0 = E$, iterate:

$$X_{k+1} = \{x | f(x, d(x)) \subset X_k\} \cup X_k$$

Until $X_{k+1} = X_k$.

A state $x \in X$ is stable with respect to E if and only if every primary cycle in the reachability set of x includes at least one state in E. A system is stable with respect to E if and only if every primary cycle of X includes at least one state in E.

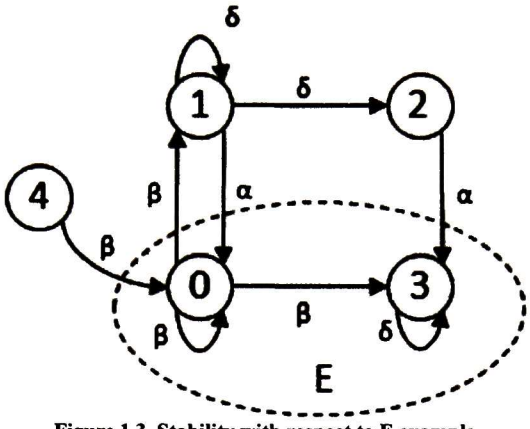


Figure 1.3. Stability with respect to E example

On the example of figure 1.3, states 0,2,3 and 4 are pre-stable with respect to $E = \{0,3\}$ but only states 2 and 3 are stable with respect to E. The state 0 is not stable because state 1 has a self-loop, this is a cycle with no states in E, and state 1 is reachable from state 0.

The property presented by Özveren does match our notion of a stable system. Also it has the advantage of being formally defined on finite automata. However, the authors do not consider Petri nets in their study.

1.2.6. ATTRACTOR STATE SETS

Brave and Heymann [Brave, 1990] studied the ability of a discrete event process to reach a set of target states from any arbitrary initial state and then remaining there indefinitely.

Let Σ be a finite alphabet. A process over Σ is modeled as a finite directed graph $G = (V, E)$ where V is the set of states and $E \subseteq V \times \Sigma \times V$ is the set of edges. Then an edge is a triple $e = (v, \sigma, u)$ where v is the start state, u is the end state and σ is the event associated to e.

Given a state set $A \subseteq V$ and an edge set $E' \subseteq E$. A is *E'-invariant* iff:

$$(\forall (v, \sigma, u) \in E')\ v \in A \rightarrow u \in A$$

Let $G = (V, E)$ be a process and let $A, B \subseteq V$ such that $\emptyset \neq A \subseteq B$. A is a strong attractor for B with respect to G iff the following conditions are satisfied:

1) A is E'-invariant.
2) For each state v reachable from B there is a path that starts at v and ends in A.
3) There are no cycles of G in $RG(B) \backslash -A$ ($RG(B)$ are the states reachable from B)

This notion is very similar to the one presented in [Özveren, 1991]; the only difference is that once the system reaches the set A it stays there indefinitely, contrary to the concept presented by Özveren et al. where the system can deviate from the desired states again.

1.2.7. CLOSURE AND CONVERGENCE: A BASIS FOR FAULT TOLERANCE

According to Arora and Gouda [1993], self-stabilization is related to fault tolerance; however self-stabilizing systems have been designed to tolerate transient faults only whereas they can be designed to tolerate a variety of fault types. The authors considered that there is a need for a uniform definition of fault tolerance that is independent of technology, architecture and application. For this reason they introduce the notions of closure and convergence.

Observe that a well-established method for verification of systems is to define a predicate (a Boolean expression) that is true during the desired or "legal" behavior. This predicate S identifies the legal system states. Also, faults of many types can be represented as actions or events perturbing the system state.

The authors require that there exists a predicate T that is weaker than S and holds during the execution of system or faults actions (T holds even in illegal states). This is, starting from a legal state (a state where both S and T are true) when a fault occurs, the next state reached must at least hold T. This constitutes the closure condition for fault tolerance.

Once the fault events stop occurring, the system can be considered recovered only if it is restored to a state where S holds. Therefore is required that every fault-free system execution upon starting from any state where T holds, eventually reaches a state where S holds. This requirement constitutes the convergence condition.

While these properties may seem similar to those presented by Özveren et al., the convergence requirement is weaker in the sense that the system is expected to return to a desired behavior once fault events stop occurring. Although the property presented by Özveren et al., is more general, it is also more restrictive in the sense that convergence should be guaranteed regardless of faults continuing to occur. Then, the concepts of stabilization presented by Arora and Gouda seem to be more appropriate for fault tolerance analysis of real-life systems.

1.2.8. GENERAL SELF-STABILIZATION OF TRANSIENT FAULTS

Marco Schneider [1993] follows the concepts presented in [Arora and Gouda, 1993]. The author presents a generalization of the concept of self-stabilization. This concept is defined for a system S with respect to predicates P and Q over its set of global states. S satisfies $Q \rightsquigarrow P$ (read as Q stabilizes to P) if it satisfies the following two properties:

1) Closure: P is closed under the execution of S. That is, once P is established in S it cannot be falsified.
2) Convergence: Starting from any global state satisfying Q, S is guaranteed to reach a global state satisfying P within a finite number of state transitions.

According to [Schneider, 1993] if S is self-stabilizing with respect to P this may be restated as $true \rightsquigarrow P$ in S. The model of failure that is discussed in this proposal considers transient failures,

which may change the global state of a system by corrupting the local state of a process or by corrupting message channels or shared memory. This proposal is similar to that of Arora et al.

1.2.9.P-CONVERGENCE

Astuti and McCarragher [1995][1996a][1996b] present a convergence concept using Markov Chains. It is based on the Stability notion proposed by [Brave,Heymann 90], yet, according to the authors, is less restrictive. Such concept is called *P-Convergence*. First, the notion of ***weak stability*** from [Brave,Heymann 90] is introduced using a probabilistic framework:

Let $G = (M, E)$ be a DES, $M_0 \subset M$, and S be a Discrete event controller for G. Then $M \xrightarrow{S/G} M_0$ (G weakly converges to M_0) if

i. M_0 is E^S-invariant (sequences of events starting in M_0 end in M_0 on the closed loop system)
ii. There exists $n \geq 1$ such that for every $q_i \in M$:

$$P^S = (q(n) \in M_0 \,|q_0 = q_i\,) = 1$$

That is, the probability of reaching M_0 in a finite number of steps n, starting from state q_i, is equal to 1 (its certain) for every $q_i \in M$. In other words, there is a guarantee that starting from any state in M , the system will reach M_0 in a finite number of steps.

Then, P-Convergence is defined, based on the notion described above:

Let $G = (M, E)$ be a DES, $M_0 \subset M$, and S be a Discrete event controller for G. Then $M \xrightarrow{S/G,p} M_0$ (G is *P-Convergent* to M_0) if

i. M_0 is E^S-invariant (sequences of events starting in M_0 end in M_0 on the closed loop system)
ii. There exists $n \geq 1$ such that for every $q_i \in M$:

$$P^S = (q(n) \in M_0 \,|q_0 = q_i\,) \geq p$$

This concept is useful for the analysis of robustness of DES when a successful return to the desired behavior is not guaranteed but it can be argued that the recovery is the most likely outcome. This concept is less restrictive than those presented by Özveren et. al. and as such is well suited for the analysis of real-life systems. However, the stability concepts presented in [Özveren 1991][Brave 1990] are stronger because they imply that the return to the desired or normal behavior is guaranteed.

1.3.FAULT TOLERANCE

Other very important problem that arises when studying the fault tolerant capabilities of DES, is determining whether a system can become fault tolerant with the help of control procedures. In this subsection, the main results dealing with this problem from the stabilization framework are reviewed.

1.3.1. STABILIZABILITY OF FINITE AUTOMATA

In [Özveren, 1991], a system is said to be stabilizable when there exists a control law such that when the system is working in closed-loop over that control law it becomes stable. In order to consider controllable inputs, the automaton definition is extended as follows :

$$G = (X, \Sigma, U)$$

Where U is the set of admissible control inputs. The system is controlled by disabling certain controllable events. The system dynamic is given by:

$$x[k+1] \in f(x[k], \sigma[k+1])$$
$$\sigma[k+1] \in (d(x[k]) \cap u[k]) \cup e(x[k])$$

Where $u[k] \in U$ is the control input after the kth-event, $e: X \rightarrow 2^{\Sigma}$ specifies the set of events which cannot be prevented to occur in a given state, and $d(x)$ represents every event that could occur in some state x.

A state feedback law is a mapping $K: X \rightarrow U$ that determines which controllable events are enabled at a certain state.

A state $x \in X$ is said to be prestabilizable with respect to $E \subset X$ if there exists a state feedback law K such that x is prestable with respect to E. A system is prestabilizable if every state of the system holds this condition.

According to [Özveren, 1991] there exists a maximal set of states which are prestabilizable with respect to E and this set is denoted by (E) . The following algorithm calculates $P(E)$:

Algorithm 1.2: find_P(E)

Input: Automaton G
Output: State feedback law K, maximal set of prestabilizable states.

Let $X_0 = E$, iterate:

1) $P_{k+1} = \{x | e(x) \neq \emptyset \text{ and } f(x, e(x)) \subset X_k \text{ or } e(x) = \emptyset \text{ and } \exists \sigma \in d(x) \text{ such that } f(x, \sigma) \subset X_k\}$

2) $K(x) = \{\emptyset \text{ if } e(x) \neq \emptyset \text{ or some } \sigma \text{ such that } f(x, \sigma) \subset X_k \text{ in other case}\}$

3) $X_{k+1} = X_k \cup P_{k+1}$

Until $X_{k+1} = X_k$

The obtained state feedback law is maximally restrictive in the sense that activates the least amount of events. The authors also show how to obtain a maximally permissive law.

A state $x \in X$ is said to be stabilizable with respect to $E \subset X$ if there exists a state feedback law such that x is stable with respect to E. A system is stabilizable if this condition holds true at every state.

In order to find the maximal set of stabilizable states ($S(E)$), $P(E)$ has to be obtained first. If every event starting from a state inside $P(E)$ would evolve into another state of $P(E)$, the calculation would be done. However, in general it is possible to have event trajectories which lead the system out of $P(E)$. As every element of $P(E)$ is prestabilizable, those problematic trajectories must start from a state in E. Then, E has to be reduced to a new set E' that does not contain these states. However it is possible that some states in $P(E)$ that were prestabilizable with respect to E are not so with respect to E'. For this reason, the maximal prestabilizable set must be calculated again.

A subset Q of X is said to be (f,u)-invariant if there exists a state feedback law K such that any event occurring from a state in Q leads the system into another state of Q under closed loop with K. Given a state set Y there exists a maximal subset of Y which is (f,u)-invariant, and is denoted by $T(Y)$.

A subset Q of X is sustainable (f,u)-invariant if it is (f,u)-invariant and every state of Q is live. Given a set Q, $I(Q)$ denotes the maximal subset of Q which is sustainable (f,u)-invariant.

The next algorithm determines the maximal subset of Q which is (f,u)-invariant:

Algorithm 1.3: findI(Q)

Input: state set Q
Output : I(Q)

Let $X_0 = Q$, iterate:

$$X_{k+1} = \{x \in X_k | exists\ \sigma \in d(x)\ such\ that\ f(x,\sigma) \subset X_k\}$$

Until $X_{k+1} = X_k$

Finally, the algorithm to find $S(E)$ is presented:

Algorithm 1.4: findS(E)

Input: Automata G.
Output: S(E)

Let $X_0 = X$, iterate:

$$X_{k+1} = I(P(E \cap X_k))$$

Until $X_{k+1} = X_k$

1.3.2.A FRAMEWORK FOR FAULT RECOVERY ON FINITE AUTOMATA

An application of the state stability concepts presented by [Özveren 1991] and [Brave 1990] is presented in [Wen 2007], [Wen, 2008], [Wen, 2009]. In this work, supervisory control is used not only to enforce a control specification, but also to ensure that a correct recovery takes place after a fault occurrence. The authors understand recovery as a return to a state which is equivalent to a legal behavior in a bounded number of steps.

Initially it is assumed that a model G (finite automaton) describes the whole behavior of a plant and a model G^N (which is a subgraph of G) represents the legal behavior only. The notion of fault tolerance characterized by [Wen 2007], [Wen, 2008], [Wen 2009] consists on a return to a non-faulty state. This is based on state stability. Also, a notion of weak fault tolerance is introduced; this does not require a return to a desired state, rather the normal behavior of the system can continue from a faulty state, keeping in mind that the performance is not going to be optimal but tolerable.

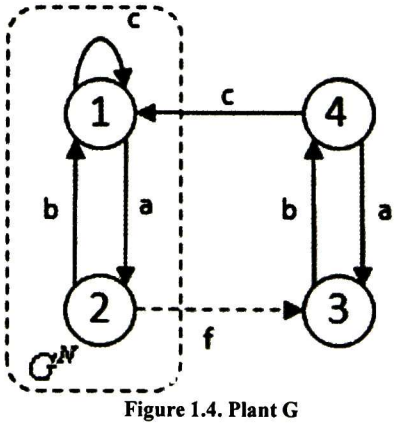


Figure 1.4. Plant G

On figure 1.6, the depicted automaton has states 1 and 2 defined as the legal or desired behavior states. There is also an event f that can occur from state 2 that represents a fault. As it can be seen there is no guarantee that the system will return to states 1 or 2 once the system leaves such states; however, the behavior after the fault is the same than that before faults, so this system is an example of a weakly fault tolerant system.
In [Wen 2008], sufficient and necessary conditions that guarantee the existence of supervisors for these notions are introduced. This work was based on finite automata.

1.3.3.STABILIZING CONTROLLERS ON FINITE AUTOMATA

In [Khatab 2002] fault recovery is studied on systems modeled as finite automata. A fault is an incontrollable event that drives the system out of the desired behavior.

According to the authors, it is assumed that two modes of operation are possible in the system: the nominal operation mode NOM and the degraded operation mode DOM. The NOM characterizes the desired behavior of the system while the DOM represents an acceptable behavior of the system after the occurrence of a permanent fault.

Two fault recovery techniques are proposed. The first consists on synthesizing a state feedback controller that guides the system back to its nominal operation mode. This procedure is strongly linked to the concept of stabilization. If it is not possible for the system to continue exhibiting the nominal behavior, the second technique is to guide the system to a degraded operational mode by restricting its behavior by means of control.

1.3.4. ACTIVE FAULT TOLERANT CONTROL USING ONLINE DIAGNOSIS

Paoli et al. consider the problem of fault tolerant control of DES modeled as automata [Paoli 2008][Paoli 2011]. In their work they present an integrated solution including fault diagnosis and recovery. They focus on permanent faults that alter the normal functioning of a system so that after a fault occurrence, a degraded behavior must be tolerated.

A central notion in the proposal of Paoli et al. is the concept of safe diagnosable DES; this refers to the fact that the faults of a given DES can be diagnosed and isolated before an illegal or irrecoverable event (from a previously defined set) occurs. A safe diagnosable DES is also safe controllable when the system can be steered away from the occurrence of illegal events after a fault event by means of control.

In [Paoli 2011] the authors present necessary and sufficient conditions for these notions along with methods to find fault tolerant controllers.

1.3.5. REDUNDANCY BASED CONTROLLER RECONFIGURATION

M. Alcaraz et al. introduced a notion of fault recovery relying on system hardware redundancies [Alcaraz 2006][Alcaraz 2007]. The authors present a method to identify hardware redundancies on Petri net modeled systems and an algebraic technique that is used to reconfigure the controller of the system when permanent faults occur.

These procedures are designed for a class of Petri nets that have a defined structure. More specifically, the models are composed by a plant and a specification sub models which are composed between them as a closed loop system, according to the theory of output regulation control presented in [Santoyo 2008].

In this proposal, the authors model independent plant processes as state machines with resource places (SMR). Basically, when a resource is not available due to a fault, the presented procedure works by searching and identifying component redundancies and then reconfiguring the controller to use these redundancies.

In this chapter the most important works on stability for DES were reviewed. While some of these approaches are adapted from continuous systems theory, the notions that are the most relevant for this research are those related to the convergence of a discrete event system to a defined state set representing the desired behavior. In this way, if faults or exceptions are of uncommon occurrence, the system would exhibit correct behavior most of the time. The concepts and background required for the study of stability on Petri nets are presented in the next chapter.

Chapter 2 BACKGROUND

In this chapter, the basic notions of the Petri net formalism as well as a well-known extension: Interpreted Petri nets are introduced. Also, the branching process representation of a Petri net is explained.

2.1.PETRI NETS

In addition to finite automata, there are other formalisms that have been widely used on the task of modeling and analyzing discrete event systems. Among them, Petri nets stand out as a powerful graphical tool that unlike finite automata, allows a clear and compact representation of complex behavior like concurrency, parallelism, synchronizations, etc.

2.1.1.DEFINITION

Definition 2.1. A Petri net structure G is a bipartite graph represented by the 4-tuple $G = (P, T, I, O)$ where:

$P = \{p_1, p_2, \dots p_n\}$ and $T = \{t_1, t_2, \dots t_m\}$ are finite sets of nodes referred as places and transitions respectively.

$I(O): P \times T \rightarrow Z^+$ is a function that represent the weighted arcs going from places to transitions (transitions to places), where Z^+ are the nonnegative integers.

Pictorially, places are represented by circles and transitions by bars, as shown in figure 2.1. The places are p1 and p2 , transitions are t1 and t2. For this example, it can be seen that I(p1,t1)=1 (there is a directed arc from p1 to t1), O(p2,t1)=1 (there is a directed arc from t1 to p2) but I(p2,t1)=0 and so on.

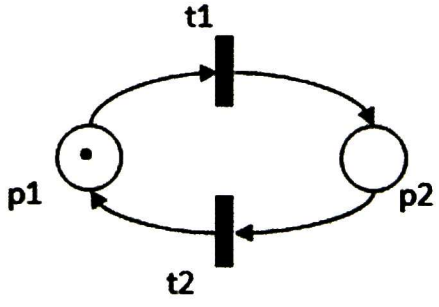


Figure 2.1: A simple Petri net

Notation. • t_j denotes the set of places p_i such that $I(p_i,t_j)>0$ and t_j • denotes the set of places p_i such that $O(p_i,t_j)>0$. Similarly, • p_i denotes the set of transitions t_j such that $O(p_i,t_j)>0$ and p_i • denotes the set of transitions t_j such that $I(p_i,t_j)>0$.

For the PN in figure 2.1, $\bullet t1 = \{p1\}$, $t1 \bullet = \{p2\}$, also, $\bullet p1 = \{t2\}$, $p1 \bullet = \{t1\}$.

Definition 2.2. The incidence matrix of G is $C = [c_{ij}]$, where $c_{ij} = O(p_i,t_j) - I(p_i,t_j)$. The marking function $M: P \rightarrow Z^+$ is a mapping from each place to the nonnegative integers, which represents the number of tokens on each place. The marking of the net on fig 2.1 can be expressed with the vector [1 0] where the first entry shows the marking of place p1 and the second entry refers to the marking of place p2.

Definition 2.3. A Petri Net system (PN) is a tuple (G, M_0) where G is a PN structure and M_0 is the *initial marking*. The transition t_j is enabled at marking M_k if $\forall p_i \in P, M_k(p_i) \geq I(p_i,t_j)$, the firing of this transition will make the system reach a new token distribution given by $M_{k+1} = M_k + Cv_k$, where $v_k(i) = 0, i \neq j, v_k(j) = 1$.

Definition 2.4. The Reachability set $R(G, M_0)$ of a PN is the set of every possible marking reachable from M_0 when firing enabled transitions only. The reachability graph is a graphical representation of the reachability set where every reachable marking is represented by a node and there exists an arc from node A to node B if and only if the marking represented by node A enables a transition T such that after the firing of T, the marking represented by node B is reached. The reachability graph for the net of figure 2.1 is shown on figure 2.2.

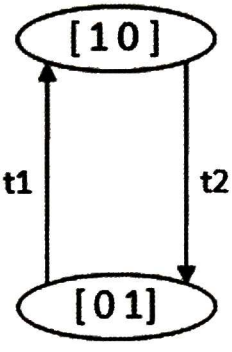


Figure 2.2: Petri net Reachability graph

Definition 2.5. Let $G = (P,T,I,O)$ be a PN structure. The induced subnet given by X, $X \subseteq P$ and denoted as [X], is the structure $[X] = (X, T^0, I^0, O^0)$ where $I^0 \subseteq I, O^0 \subseteq O$ such that $I^0: X \times T^0 \cap I, O^0: X \times T^0 \cap O$, and $T^0 = \bullet X \cup X \bullet$.Similarly, the induced subnet given by Y, $Y \subseteq T$ and denoted as [Y] is the structure $[Y] = (P^0, Y, I^0, O^0)$ where $I^0 \subseteq I, O^0 \subseteq O$ such that $I^0: P^0 \times Y \cap I, O^0: P^0 \times Y \cap O$, y $P^0 = \bullet Y \cup Y \bullet$. On figure 2.3 a Petri net is depicted on the left while the subnet induced by {t1,t2}is depicted on the right.

Definition 2.6. Let (G, M_0) be a PN system where $G = (P,T,I,O)$. If $|\bullet t_j| = |t_j \bullet| = 1, \forall t_j \in T$, then (G, M_0) is a state machine. If $|\bullet p_i| = |p_i \bullet| = 1, \forall p_i \in P$, then (G, M_0) is a marked graph.

If (G, M_0) is a marked graph and a state machine simultaneously, then it is a simple cycle. The net on figure 2.1 is both a state machine and a marked graph, so it is a simple cycle.

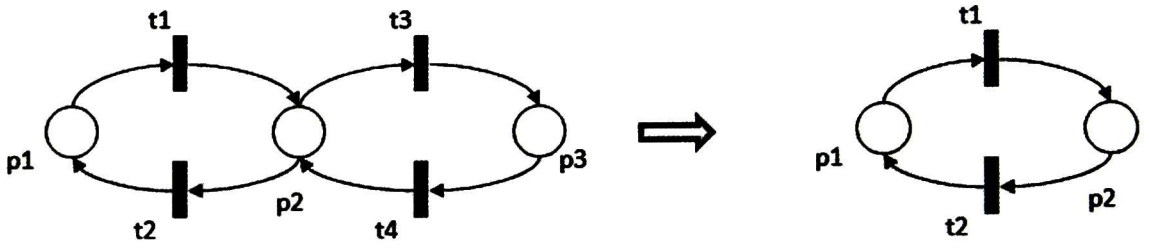


Figure 2.3: PN structure and one of its subnets

Definition 2.7. Let (G, M_0) be a PN system. A P-invariant Y (T-invariant X) of a PN is a rational solution to the equation $Y^T C = 0$ $(CX = 0)$. An invariant is said to be semipositive if every entry on the vector is greater or equal than zero. The support of a vector Y denoted as $< Y >$ represents a set of transitions or places and is defined as $< Y >= \{y_i \mid Y[i] > 0\}$. Let Y be a P-invariant (T-invariant X), $[< Y >]$ ($[< X >]$) is called P-Component (T-Component). On figure 2.3 there is only one minimal P-Invariant, which support is {p1,p2,p3} and there are two minimal T-Invariant with supports {t1,t2} and {t3,t4} respectively. The subnet appearing on the right of figure 2.3 is a T-Component.

2.1.2. INTERPRETED PETRI NETS

Like finite automata, Petri nets can be extended to take into account the existence of controllable and incontrollable events and to distinguish between the different input and output events. This extension is generally known as Interpreted Petri nets (IPN) and its definition is detailed next.

Definition 2.8. An IPN is a 4-tuple $Q = (N, \Sigma, \lambda, \varphi)$ where:

$N = (G, M_0)$ is a PN system.
$\Sigma = \{\alpha_1, \alpha_2, \ldots \alpha_r\}$ is the input symbol alphabet, where α_i is an input symbol.
$\lambda: T \rightarrow \Sigma \cup \{\epsilon\}$ is the transition labeling function with the following restriction:
$\forall t_j, t_k \in T, j \neq k$, if $\forall p_i, I(p_i, t_k) = I(p_i, t_j) \neq 0$ and $\lambda(t_k) \neq \epsilon, \lambda(t_j) \neq \epsilon$, then $\lambda(t_k) \neq \lambda(t_j)$.
$\varphi: R(Q, M_0) \rightarrow \{Z^+\}^q$ is the output function which maps an output vector to every reachable marking and q is the number of outputs available on the places.

If $\lambda(t_i) \neq \epsilon$, the transition t_i is said to be controllable. Otherwise, it is uncontrollable. On Interpreted Petri nets any uncontrollable event is represented by ϵ.

A place p_i is measurable if the ith column vector of matrix φ is different from null, otherwise is not measurable.

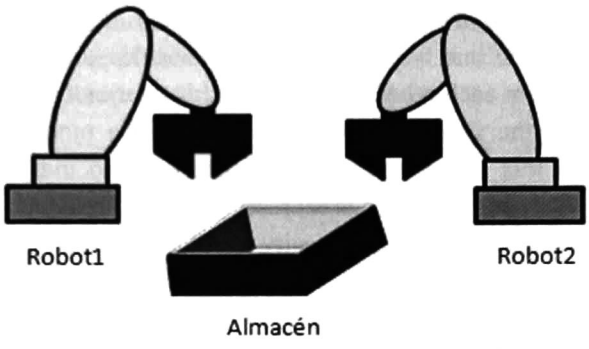


Figure 2.4: Discrete event system example

A small system with two robotic arms and a storage box is depicted in figure 2.4. One of the robotic arm handles the load operation while the other handles the unload operation, these events are controllable. The corresponding IPN model is shown on figure 2.5. There is a place that represents the storage box occupation which is measurable. Also, there are two "working" places that represent that there is a load or unload operation taking place respectively, the events of finishing an operation are uncontrollable.

Figure 2.5: PN model for system on fig. 2.4

However, the model presented does not have any means of control, so, the storage box could be overloaded and a collision between the robotic arms is possible. Figure 2.6 shows the same system in closed-loop (controlled), where the control restricts the load and unload operation to avoid going over the storage capacity (3 objects) and avoid collisions.

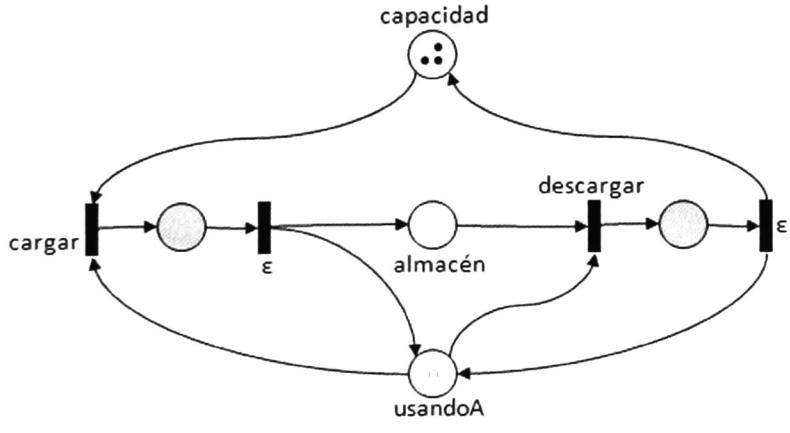


Figure 2.6: Controlled system model

2.1.3. PETRI NET MODEL COMPOSITIONS

In DES models it is common that two or more state variables are related; their behavior is not completely independent from each other. In Petri nets this relation can be represented by merging the individual models of the state variables; this results in a model which shows how those variables are related. In this subsection, two different ways to merge nets are presented. An example of a system with many variables is represented on figures 2.4-2.6 which is composed of two state variables: the remaining storage capacity and the availability to do some action over the storage area.

When two state variables have transitions which represent the occurrence of the same event, there is a synchronic relation or synchronization between these variables. In figure 7 the state variables of the load-unload system are shown. It can be seen that there are transitions representing the same event in both models. The complete model that results from the synchronic composition is the model on figure 2.6 (symplified, some duplicated non-measurable places were deleted).

A permission relation exists between two state variables A, B when there is some event that changes the value of variable A but this event needs that variable B has a specific value to occur, and this occurrence does not modify the value of variable B at all. A very simple system is depicted in figure 2.8. There is a liquid container having only two measurable states, empty and full and can go from one state to the other only when the valve state variable is on value "Open".

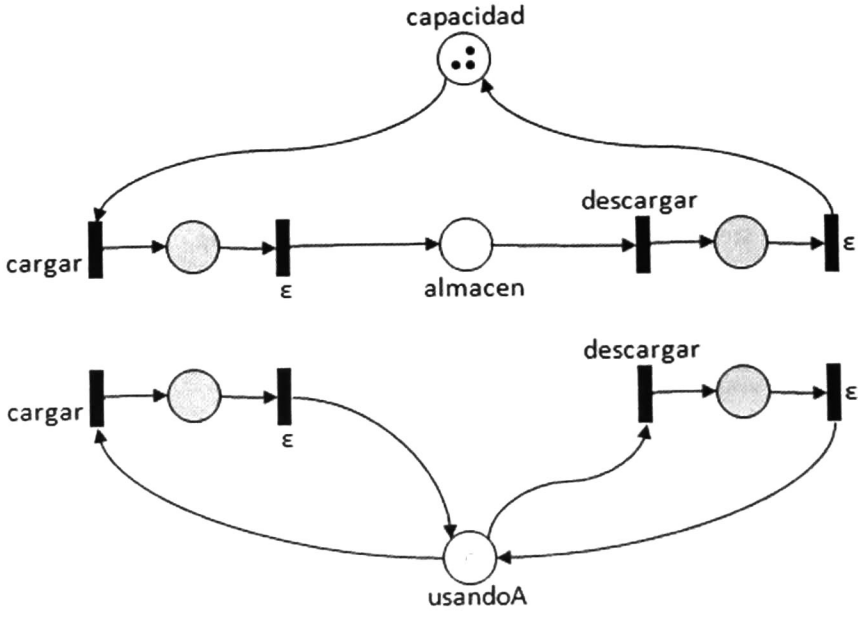


Figure 2.7: State variables for system on fig. 2.4: capacity(up) and storage use (down).

Intuitively, on a synchronic composition, the transitions representing the same event fuse in one transition. On a permission composition, where the state variable A requires the state variable B

to be in a specific state, a bidirectional arc from the transition on variable A model to the corresponding place of variable B model is included. Synchronic and permissive relations are defined below according to [Rivera, 2004], [Rivera 2005].

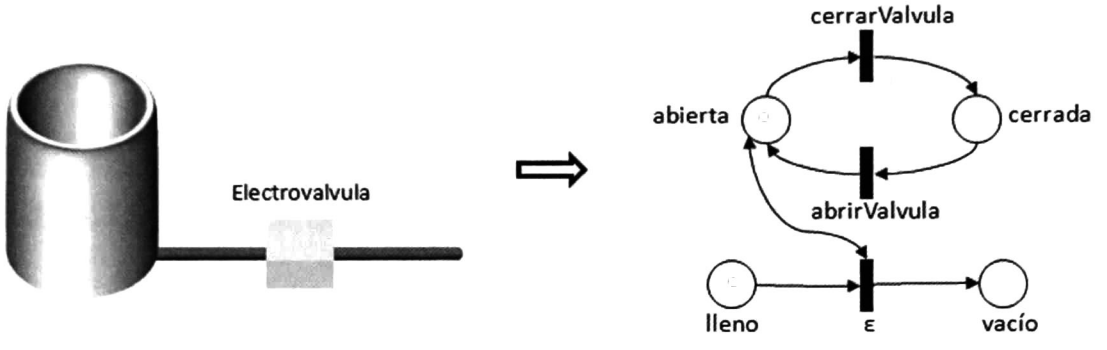


Figure 2.8: Permission relation example

Definition 2.9. Let (G, M_0) be a PN and LABELS = $\{l_1, l_2, \ldots l_w\}$ be a set of arbitrarly defined labels. (G, M_0) is a labelled PN if there exists a function tLab: $T \rightarrow LABELS$ mapping a transition to some label and a function pLab: $P \rightarrow 2^{LABELS}$ that maps a place with a set of labels.

In order to complete a synchronic composition, those transitions that represent the occurrence of the same event should have the same label. For establishing a permission composition, the places that will allow some transition t to fire when marked should have the same label than transition t.

Definition 2.10. Let (G^1, M_0^1) y (G^2, M_0^2) be two labelled PN where $LABELS_1$ is not necessarily different from $LABELS_2$. The synchronic composition of the two PN is the labelled net $(G^{1||2}, M_0^{1||2})$ given by :

$$P_{1||2} = P_1 \cup P_2$$

$$T_{1||2} = (T_1 - cT_1) \cup (T_2 - cT_2) \cup cTM_0^{1||2} = M_0^1 \cup M_0^2$$

$$I_{1||2} = I_1|_{\{(p_i,t_j)|t_j \in T_1 - cT_1\}} \cup I_2|_{\{(p_i,t_j)|t_j \in T_2 - cT_2\}} \cup$$
$$\{((p_i, t_j), 1)|(t_j \in cT_1 \wedge I_1(p_i, t_j) = 1) \vee (t_j \in cT_2 \wedge I_2(p_i, t_j) = 1)\}$$

$$O_{1||2} = O_1|_{\{(p_i,t_j)|t_j \in T_1 - cT_1\}} \cup O_2|_{\{(p_i,t_j)|t_j \in T_2 - cT_2\}} \cup$$
$$\{((p_i, t_j), 1)|(t_j \in cT_1 \wedge O_1(p_i, t_j) = 1) \vee (t_j \in cT_2 \wedge O_2(p_i, t_j) = 1)\}$$

where:

$cT_1 = \{t_k \in T_1 | \exists t_i \in T_2, tLab_1(t_k) = tLab_2(t_i)\}$,

$cT_2 = \{t_k \in T_2 | \exists t_i \in T_1, tLab_2(t_k) = tLab_1(t_i)\}$,

$cT = t_k | \exists t_i \in cT_1, \exists t_j \in cT_2, tLab_1(t_i) = tLab_2(t_j)$, y

$f|$ *is the f function restricted to some subset.*

Definition 2.11. Let (G^1, M_0^1) , (G^2, M_0^2) be two labelled PN, where $LABELS_1$ is not necessarily different from $LABELS_2$. The permissive composition of the two PN is the labelled PN $(G^{1\circ 2}, M_0^{1\circ 2})$ given by:

$$P_{1\circ 2} = P_1 \cup P_2$$

$$T_{1\circ 2} = T_1 \cup T_2$$

$$M_0^{1\circ 2} = M_0^1 \cup M_0^2$$

$$I_{1\circ 2} = I_1 \cup I_2 \cup \{((p_k, t_j), 1) | (p_k \in P_{1\circ 2}, t_j \in T_{1\circ 2}) \wedge (tLab_1(t_j) \in pLab_2(p_k) \vee tLab_2(t_j) \in pLab_1(p_k))\}$$

$$O_{1\circ 2} = O_1 \cup O_2 \cup \{((p_k, t_j), 1) | (p_k \in P_{1\circ 2}, t_j \in T_{1\circ 2}) \wedge (tLab_1(t_j) \in pLab_2(p_k) \vee tLab_2(t_j) \in pLab_1(p_k))\}$$

2.2. PETRI NET UNFOLDINGS

While complex behavior like concurrency can be expressed in a compact way with Petri nets, the actual number of states represented by the net can be huge. In other words, the number of reachable markings of a net could possibly grow exponentially with the size of the model. This is generally known as the state explosion problem, and has been one of the most important issues on Petri nets. In [McMillan, 1993] a new technique to avoid this problem has been proposed, which is based on the concept of Petri net unfoldings. This method is summarised in the next subsections.

2.2.1. OCCURRENCE NETS

The following notions have been taken mainly from [Esparza, 2002]. First, the causal, conflict and concurrency relations between nodes of a net are recalled.

Definition 2.12. Two nodes x, y are in ***causal relation***, denoted by $x < y$, if the net contains a path from x to y; they are in ***conflict relation***, denoted by $x\#y$, if the net contains two paths $st_1 \ldots x$ and $st_2 \ldots y$ starting at the same place s, and such that $t_1 \neq t_2$; they are in ***concurrency relation***, denoted by $x\ co\ y$, if neither $x < y$ nor $y < x$ nor $x\#y$.

These relations are shown in figure 2.9. In 2.9.a the example net shows that there exists a directed path between places A and B; this means that A and B are in a causal relation. Next, on 2.9.b places A and B are in a conflict relation because they are on mutually excluding execution paths starting from the same place. Finally, on 2.9.c places A and B are concurrent.

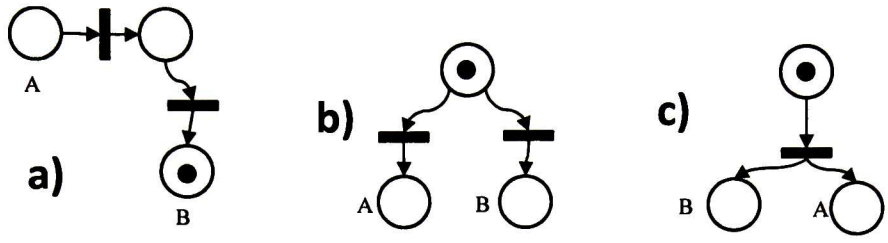


Figure 2.9. a) Causal relation. b) Conflict relation. c) Concurrent relation.

Definition 2.13. An *occurrence net* is a net $ON = (B, E, I, O)$ such that:

1) $\forall b \in B, |\bullet (b)| \leq 1$;
2) ON is acyclic;
3) $\forall x \in B \cup E$, the set $\{y \in B \cup E | \, y < x\}$ is finite;
4) No element is in conflict with itself.

where B and E are called *conditions* and *events* respectively; *Min*(ON) denotes the set of minimal elements of $B \cup E$ that have an empty preset.
In words, an occurrence net is a Petri net model which has a simple, tree-like structure. It is easy to see that any two nodes of an occurrence net are either in causal, conflict, or concurrency relation. An 1-bounded Petri net can be rewritten as an occurrence net by a process called *unfolding*. The resulting nets are called *branching processes.*

2.2.2. BRANCHING PROCESSES

Definition 2.14. A *branching process* of a net system $\Sigma = (P, T, I, O, M_0)$ is a labelled occurrence net $\beta = (B, E, I_2, O_2, \rho)$, where the labelling function ρ satisfies the following properties:

1) $\rho(B) \subseteq P$ *and* $\rho(E) \subseteq T$ (ρ preserves the nature of nodes);
2) For every $e \in E$, the restriction of ρ to $\bullet e$ is a bijection between $\bullet e$ (in β) y $\bullet \rho(e)$ (in $\sum$), and similarly for $e \bullet$ and $\rho(e) \bullet$ (ρ preserves the environments of transitions);
3) The restriction of ρ to $Min(O)$ is a bijection between $Min(ON)$ and M_0 (β *starts* at M_0);
4) For every $e_1, e_2 \in E$, if $\bullet e_1 = \bullet e_2$ and $\rho(e_1) = \rho(e_2)$ then $e_1 = e_2$ (β does not duplicate the transitions of Σ).

A *branching process (BP)* is a representation of a PN behavior which has a simpler structure than the original net, and smaller size than *R*(*G, M0*). When the PN describes repetitive behaviour, as in figure 2.10, it has an infinite BP which is obtained by unfolding as much as possible and is called *the unfolding* of the system. This means that branching processes of the same system differ on how much they unfold; so it is natural to introduce a prefix relation between BP's.

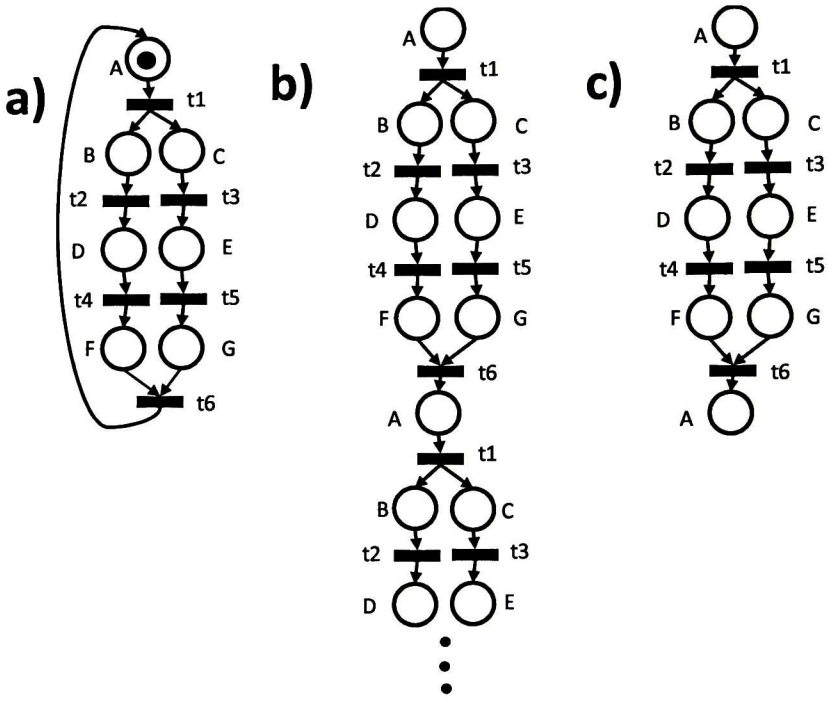


Figure 2.10: Petri net and two of its branching processes

Definition 2.15. Let $\beta' = (ON', \rho')$ and $\beta = (ON, \rho)$ be two branching processes of a net system. β' is a *prefix* of β if ON' is a subnet of β satisfying:

$Min(ON)$ belongs to ON'
If a condition b belongs to ON', then its input event $e \in \bullet b$ in ON also belongs to ON' (if it exists)
If an event e belongs to ON', then its input and output conditions $\bullet e \cup e \bullet$ in ON also belong to ON'
ρ' is the restriction of ρ to ON'.

The branching process of figure 2.10.c is a prefix of the branching process on 2.10.b.

2.2.3. CONFIGURATIONS

Definition 2.16. A *configuration* C of a BP is a set of events satisfying the following two conditions:

1) $If\ e \in C \Longrightarrow \forall e' \leq e : e' \in C$ (C is causally closed).
2) $\forall e, e' \in C: \neg(e \# e')$ (C is conflict free).

Intuitively, a configuration is a set of transitions which can occur from the initial marking in some order. On figure 2.10.c, the set labelled with {t1,t2,t3} is a configuration but {t2,t3} is not (the set is not casually closed).

Definition 2.17. *Mark*(C) is the reached marking by firing the transitions in C. $C \oplus E$ is called the *extension* of C if $C \cup E$ is a configuration s.t. $C \cap E = \emptyset$. E is called a *suffix* of C.

2.2.4. FINITE COMPLETE PREFIXES

In this subsection the method for the building of the unfolding is presented along the procedure for the construction of a finite branching process which represents every reachable marking.

The unfolding of a net system Σ is going to be represented with a set of nodes $\{n_1, \dots n_k\}$. A node is either a condition (place) or an event (transition). The condition node is a pair (s, e) that includes a place s and its only input transition e (every place has only one input transition, as in occurrence nets). An event is also a pair (t, X) where t is a transition of Σ and X represents the set of input places to t.

Definition 2.18. A co-set is a set of places in a branching process which are pairwise in concurrent relation.

Definition 2.19. Let β be a branching process of a net system Σ. The possible extensions of β are the pairs (t, X) where X is a co-set of conditions of β and t is a transition of Σ such that:

$\rho(X) = {}^\bullet t$ and
(t, X) does not already belong to β

The algorithm which builds the unfolding of a net system is presented next.

Algorithm 2.1 The unfolding algorithm

Input: A net system $\Sigma = (N, M_0)$ with $M_0 = \{p_1, \dots p_n\}$.
Output: The unfolding *Unf* of Σ
begin
Unf← $\{(p_1, \emptyset), \dots, (p_n, \emptyset)\}$;
pe←*PE(Unf);//PE(Unf) are the possible extensions of Unf*
while *pe*≠ ∅ do
    add to *Unf* an event $e = (t, X)$ of *pe* and a condition (s, e)
        for every output place s of t;
    pe←*PE(Unf);*
endwhile
end

For net systems with cyclic behavior, the algorithm above never terminates. However, in order to analyze some properties of the system a finite branching process is needed. For this reason it is necessary to construct a finite initial part of the unfolding which represents the entire behavior of the system. Such finite initial part of the unfolding is called *finite complete prefix*.

Definition 2.20. A BP β is *complete* if $\forall M \in R(G,M_0)$ there is a configuration C in β such that:

1) $Mark(C) = M$;
2) For every transition t enabled by M there exists a configuration $C \cup \{e\}$ s.t. $e \notin C \wedge \rho(e) = t$.

Since an 1-bounded net has only finitely many reachable markings, its unfolding contains at least one complete finite prefix. McMillan had the idea of identifying certain events, called cut off events, at which the construction can be stopped without losing information. Also, McMillan attached to each event e added by the unfolding algorithm a reachable marking of Σ. For this, the local configuration $[e]$ is computed first and then e is related to the reachable marking $Mark([e])$.

Definition 2.21. The local configuration $[e]$ of an event e of a BPis the set of events e' such that $e' \leq e$. For example in figure 2.10.c [t4]={t1,t2} and Mark([t4])={F,C}.

2.2.5.ADEQUATE ORDERS

According to [Esparza, 2002] if there are events e, e' such that $([e]) = Mark([e'])$, it is sufficient to continue the construction of only one of the two (the extensions after them are isomorphic). The choice between [e] and [e'] is made on the basis of a partial order. Esparza proved that all orders satisfying the 3 properties below lead to finite complete prefixes:

Definition 2.22. A partial order $\prec$ on the finite configurations of the unfolding of a net system is an *adequate order* if:

1) $\prec$ is well-founded,
2) $C_1 \subset C_2$ implies $C_1 \prec C_2$, and
3) $\prec$ is preserved by finite extensions.

In [McMillan, 1993] the author successfully used the size of the local configuration as an ordering. It is clear that this is an adequate order.

2.2.6.CUT-OFF EVENT

The key concept used to build a finite complete prefix (FCP) is that of a *cut-off event.* Intuitively, when such an event is found, there is no need to continue expanding the branching process beyond that point, as every marking added after that transition is already represented.

Definition 2.23. Let $\prec$ be an adequate order on the configurations of the unfolding of a net system. Let β be a prefix of the unfolding containing an event e. The event e is a *cut-off event* of β (with respect to $\prec$) if β contains a local configuration $[e]$ such that:

1) $Mark([e]) = Mark([e'])$, and
2) $[e'] \prec [e]$.

In figure 2.10.c an event e labelled t1 could be added to the unfolding, however it is clear that there exists another event e' labelled t1 with the same final marking. Furthermore [e] ={t1,t2,t3,t4,t5,t6} and [e']=∅ so e' has a smaller configuration. This means that e is a cut off event and the depicted branching process is a finite complete prefix.

The number of events in the finite complete prefix depends on the adequate order used to build it; it is possible that two events with the same final marking have local configurations of equal size, so the adequate order proposed by McMillan is partial.

An adequate total order for 1-bounded nets was presented in [Esparza, 2002]. In the case of a total order, for any pair of events $[e'], [e]$ it is true that either $[e'] \prec [e]$ or $[e] \prec [e']$ so if $Mark([e]) = Mark([e'])$ either $[e']$ is a cutoff event or $[e]$ is. Then, in any finite complete prefix obtained with respect to an adequate total order which does not include the cutoff events, $Mark([e_1]) = Mark([e_2])$ only if $e_1 = e_2$. In order to present this adequate order, some concepts from [Esparza, 2002] have to be reviewed first.

Let $\Sigma = (N, M_0)$ be a net system, and let $\ll$ be an arbitrary total order on the transitions of Σ. Given a set E of events, let $\varphi(E)$ be that sequence of transitions which is ordered according to $\ll$ and contains each transition t as often as there are events in E with label t. For instance, suppose $t_1 \ll t_2 \ll t_3 \ll t_4$, and also suppose that the set E contains four events labelled by t_1, t_2, t_2 and t_3 then $\varphi(E) = t_1 t_2 t_2 t_3$ It is said that $\varphi(E_1) \ll \varphi(E_2)$ if $\varphi(E_1)$ is lexicographically smaller than $\varphi(E_2)$ with respect to the order $\ll$. Next, the *Foata Normal Form*[Diekert, 1990] of a configuration is defined.

Given a finite configuration C, its *Foata Normal Form* is the list of sets of events constructed by the following algorithm:

Algorithm 2.2. Foata Normal Form of a configuration

Input: A configuration C of a branching process
Output: The Foata normal form FC of C.

$FC \leftarrow \emptyset$;
while $C \neq \emptyset$ do
    append $Min(C)$ to FC;
    $C \leftarrow C \backslash Min(C)$;
endwhile
end

Definition 2.21. Let C_1 and C_2 be two configurations of the unfolding of a net system. $C_1 \prec_F C_2$ holds if:

1) $|C_1| < |C_2|$, or
2) $|C_1| = |C_2|$ and $\varphi(C_1) \ll \varphi(C_2)$, or
3) $|C_1| = |C_2|$, $\varphi(C_1) = \varphi(C_2)$ and $FC_1 \ll FC_2$

In words, in order to determine which configuration is smaller, the sizes of the configurations are compared first, if they have the same size then their transitions are ordered by $\ll$ and the resulting sequences are compared lexicographically. If these sequences are equal, then the Foata Normal Form of the configurations are obtained and compared. The demonstration that $\prec_F$ is indeed an adequate total order and the complete algorithm for finding a finite complete prefix is found on [Esparza, 2002].

In this chapter, fundamental definitions on Petri nets and one of its extensions were reviewed, also, a method to cope with the state explosion problem called Petri net unfoldings was presented. In the next chapter, the issue of state stability on a subclass of Petri-net-modeled DES´s is explored.

Chapter 3 STABILITY ON PETRI NETS

In this chapter the notion of stability that is going to be used in this work is introduced. Also, a technique for deciding the property in a class of Petri net models, named Closed Loop System Model (CLSM) nets is proposed.

3.1.PROBLEM STATEMENT

In the occurrence of non-prevented events that detour the operation out of normal or desired behaviour states, it may be interesting to know if it is guaranteed that the system eventually would return to the expected behaviour. This ability is referred in our study as the stability of discrete event systems.

This property has been studied in [Ozveren, 1991] for DESs modeled with finite state automata. Assuming that there is a prescribed set of "good" system states S, such that, starting from these states and in the absence of errors or anomalous events, only good event trajectories are possible, a given state X is said to be stable iff every cycle in the automaton, whose states are reachable from X, has at least one state in S. A system is stable if every state in the system is stable. In this way, if errors or anomalous events are not very common, the system would exhibit legal behaviour most of the time. Next, a similar property for IPN models is defined.

Definition 3.1. Let (Q, M_0) be an IPN system, and let $S \subseteq R(Q, M_0)$ be the set of expected markings. (Q, M_0) is stable w.r.t. S iff $\forall M_k \subseteq R(Q, M_e) s.t. M_e \in S$ it is possible to reach a marking in S only by the firing of finite length transition sequences.

This notion is illustrated in the finite automaton of figure 3.1 in which one can distinguish two subsets of markings: $S_1=\{M_0, M_1, M_2, M_3\}$ representing a normal functioning operation mode, and $S_2= \{M_6, M_7\}$ representing a degraded functioning mode; $S=S_1 \cup S_2$ is the set of expected markings. Arcs labelled with ε represent uncontrollable events that detour the system form the expected behaviour. In this system if the states M_4 or M_5 are reached, the system can back to some state in S. Instead, if M_8 or M_9 are reached, a large sequence (maybe infinite) of events may occur before returning to M_6.

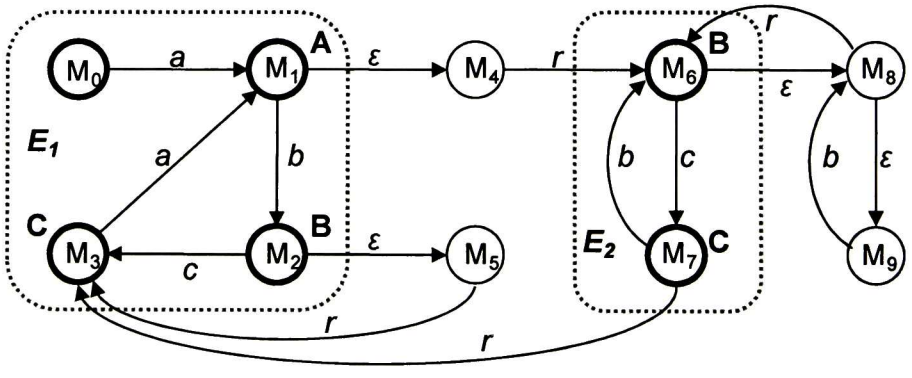


Figure 3.1 Stability notion

According to this definition, stability can be characterized in terms of the structure of the reachability graph as follows.

Proposition 3.1. Let (Q, M_0) be an IPN system, and let $S \subseteq R(Q, M_0)$ be the set of expected markings. A marking $M \in R(Q, M_0)$ is stable w.r.t. S iff every cycle in the graph representation of $R(Q, M)$ contains at least one marking in S, and every node in the graph representation of $R(Q, M)$ enables at least one event, that is, $R(Q, M)$ is nonblocking. Then, (Q, M_0) is stable w.r.t. S iff $\forall M_k \subseteq R(Q, M_0)$, M_k is stable w.r.t. S.

Proof. It is straightforward. The conditions that constraints to include a $M_e \in S$ in every cycle in the reachability graph of (Q, M_0) and the fact that there exists no dead states assure that the length of firing sequence to return to S is finite. ■

At first, it may seem that the problem of determining if a system is stable with respect to an expected state set S is the same problem as determining if S is a *home space*. A set of markings $\mathcal{M}$ is said to be a *home space* if for every $M \in R(Q, M_0)$, there exists $M_1 \in \mathcal{M}$ such that $M_1 \in R(Q, M)$ [Esparza 1994]. According to this definition, it is true that if a system is stable with respect to S, then S must be a home space. However it is *not true* that if S is a home space, then the system is stable with respect to S as can be seen in figure 3.2:

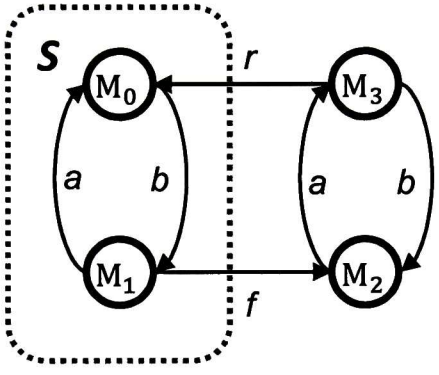


Figure 3.2. Example showing the difference between Stability and Home Space notions.

Every reachable marking in figure 3.2 can also reach either M_1 or M_0 (both in this case) this means that S is a home space. However it is not guaranteed that either M_1 or M_0 *will always be reached in a finite number of steps* as there exists a cycle of undesired markings in the graph $(M_2 M_3 M_2)$. This implies that the system is not stable with respect to S.

A procedure for implementing a stability test derived from this characterization has been presented in [Lutz-Ley, 2012a]. Although the algorithm is performed in polynomial-time on the size of $R(Q, M_0)$, in general the analysis is not efficient for large systems, because the computation of the reachability graph is required. Then, the analysis of stability from the PN structure instead of the Reachability graph is preferred; this study is performed on a class of controlled Petri net models which is detailed next.

3.2. STABILITY ON CLSM NETS

3.2.1. CLSM NETS

IPN have been widely used for modeling concurrent discrete manufacturing systems. In particular, DES's models expressed by a subclass of 1-bounded IPN are the focus of this section. In this subclass, the plant and the control specification are clearly distinguishable as components, and their interaction is clearly stated.

The plant model is formed by a set of IPN modules that describe the behaviour of the process state variables. These models are related with the controller model by means of permissive compositions in both directions. The controller places enable, when they are marked, transitions of the plant model components (through permissions). Accordingly, the places of the plant model inform to the controller that a relevant situation has been reached when they are marked.

A model built in this manner preserves the appearance of modules of both plant components and controller, and distinguishes clearly their interaction, which establishes the controller-plant closed-loop relationship. This kind of models is called Closed-Loop System Model (CLSM); this is illustrated in Figure 3.3.

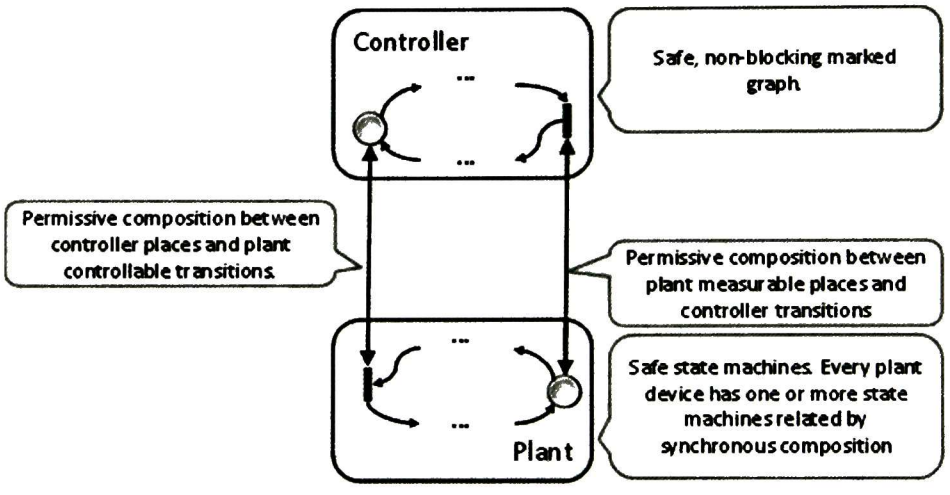


Figure 3.3 General scheme of CLSM nets

Now the IPN class CLSM is defined in terms of its components and how these components are related.

3.2.2. CONTROLLER AND PLANT COMPONENTS

The controller component is a set of synchronized repetitive sequential processes; it is recursively defined as follows.

Definition 3.2. Controller component.

A non-blocking 1-bounded IPN whose structure is a circuit is a controller component.

Two controller components synchronized through a single transition form a controller component.

Property. A controller component is, in fact, a non-blocking marked graph. This is a special case of LB-circuits compositions characterised in [Koh, 1991].

A plant component describes the behaviour of a physical device on the system. Every device can have more than one state variables, so it could be formed by one or more state machines based on the representation of its behaviour; these subcomponents are related between them by synchronous composition.

The plant component is recursively defined as follows.

Definition 3.3. Plant component.

An 1-bounded IPN whose structure is a strongly connected state machine is a plant component. Two synchronized plant components form a plant component.

The CLSM is composed by one controller component and one or several plant components related by permissive compositions.

Definition 3.4. Closed-Loop System Model (CLSM)

Let (G_c, M_c) be a controller component, with (G_{c_k}, M_{c_k}) for k=1,2...z being the set of simple circuits that compose it; let (G_e, M_e) for $e = 1,2, \ldots r$ be a collection of plant components. If $\forall e, (G_e, M_e)$ is related with (G_c, M_c) using the permissive composition, according to the following rules:

1) If (G_i, M_i) is permissive-related with (G_{c_r}, M_{c_r}) and (G_{c_u}, M_{c_u}), then $u = r$.
2) $\forall t_r \in T_c$, t_r is permissive related with at least one place from some plant component.
3) For every permissive relationship from place p_i to transition t_j:
   a) If $p_i \in P_c$, then t_j is manipulated
   b) If $p_i \in P_e$, for e=1,2,...r , then p_i is measurable.

then the resulting model is a CLSM.

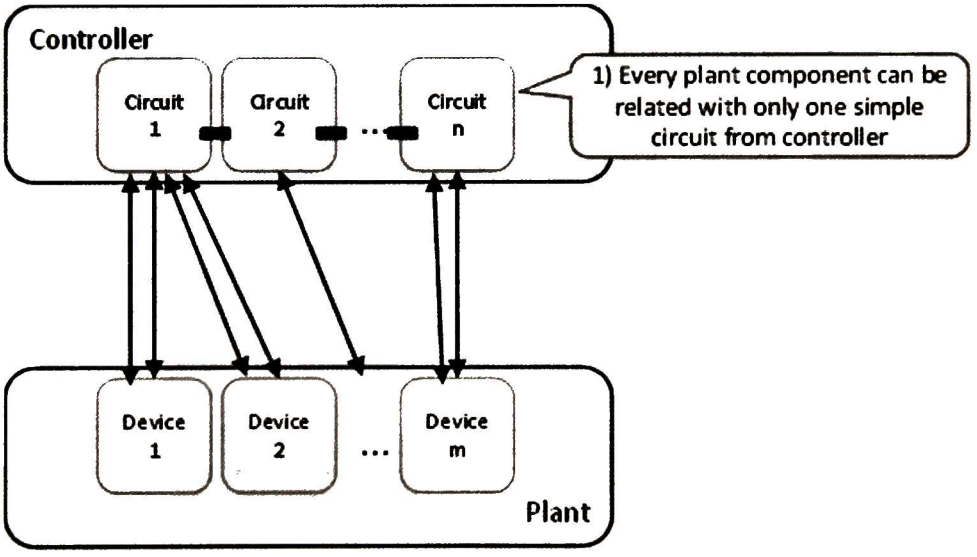


Figure 3.4. CLSM first condition

Condition 1 limits the permissions of every plant component such that they relate with only one simple circuit of the controller, allowing a simpler analysis later.

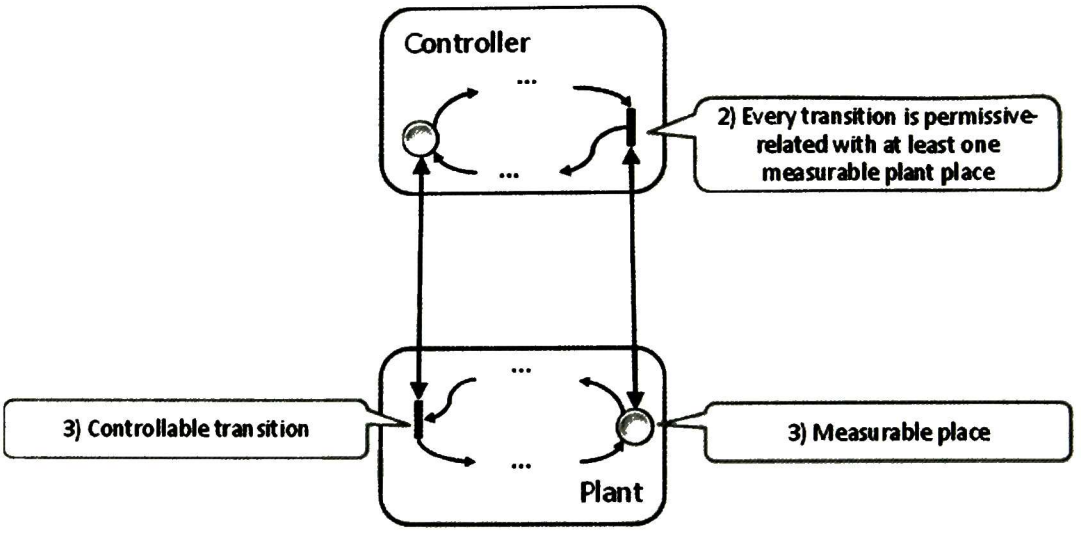


Figure 3.5. CLSM conditions 2 and 3

Condition 2 express that every transition of the controller needs permission from at least one place of a plant component; in this way, the controller goes to the next state only after the plant reaches a given state, and thus a desired output.

Conditions 3a and 3b represent basic consistency restrictions: only controllable transitions can be disabled by permissions, and only measurable places can be read by the controller component, also using arcs in both directions (permission self-loops).

In other words, in a CLSM the controller strategy can be represented by a non-blocking marked graph and each simple circuit of the controller interacts with a part of a complete process. Each of these parts is represented by one or more plant components, in which every plant component is permissive-related with only one of the simple circuits of the controller component and vice versa. The obtained model is 1-bounded, which is a property well adapted for actual manufacturing systems.

3.2.3. EXAMPLE

Consider the pick and place system showed in figure 3. The magnetic grip is used for transferring metallic parts from location 1 to location 2. The corresponding CLSM model is depicted in figure 3.7; it explicitly shows the modules and their interactions. It consists of the controller component and two plant components, one for describing the gripper displacement and position (Plant Component 2), and the other one for representing the gripper state (Plant Component 1). Permission arcs between the controller and the plant components are depicted by arcs in both directions (which are depicted as dotted bidirectional arcs), non-measurable plant places are depicted as gray circles, and uncontrollable transitions are labelled with ε.

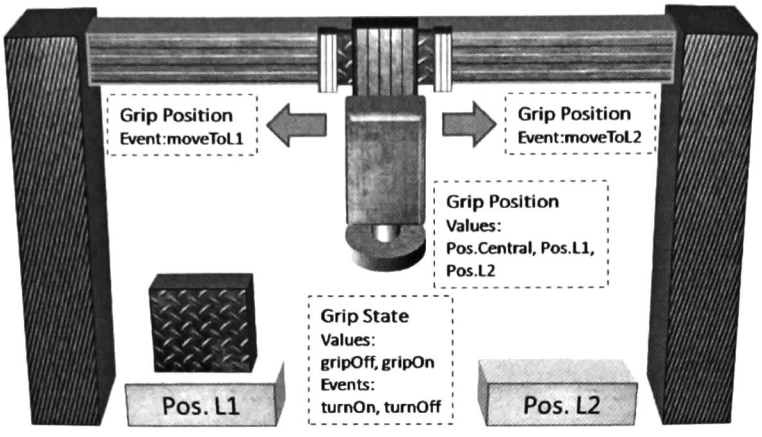


Figure 3.6. A simple pick and place system.

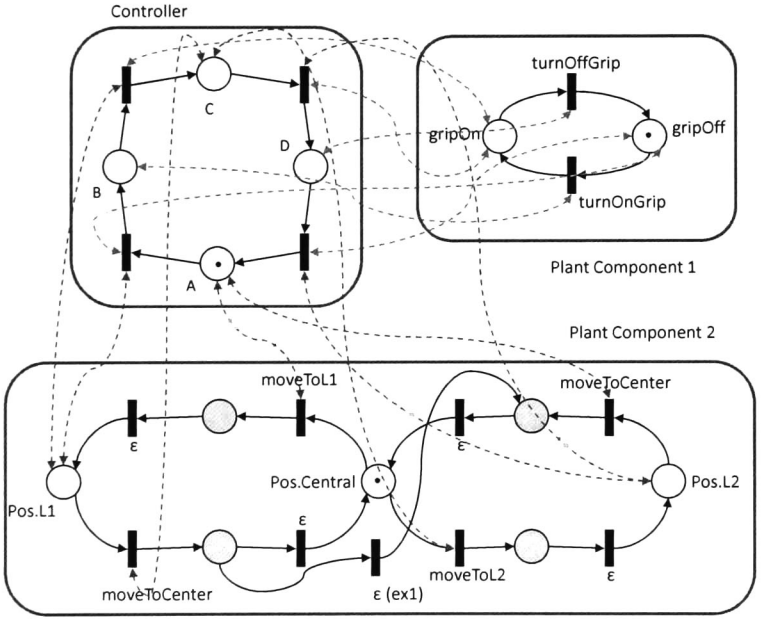


Figure 3.7. CLSM for the example of figure 3.5.

3.2.4. CHECKPOINT MARKINGS

In CLSM, the controller places enable transitions in the plant components, and then these components evolve until the marked places inform to the controller a given situation (feedback) which is taken into account to advance the controller operation.

Definition 3.5. The markings that enable the transitions of the controller component (Q_c, M_c) are called *checkpoint markings.* They are composed by the marking of (Q_c, M_c) and the markings of the plant components (G_{c_k}, M_{c_k}).

Checkpoint markings represent the set of expected markings S. This set can be obtained from a CLSM (Q, M_0) by inspection of those places in ${}^{\bullet}t_j$, $\forall t_j \in T_c$ of (Q_c, M_c) (places of the controller component), which are related by permissive composition with the plant components. For example, consider the CLSM in figure 3.7; for the controller transition whose input place is A, the preconditions that allow the firing of the transition are {A, PosL1, gripOff} and for this reason the marking {A, PosL1, gripOff} is in the set of checkpoint markings, which are showed in figure 3.8 in the form of a graph that represent the output evolution of the plant to complete a correct cyclic operation.

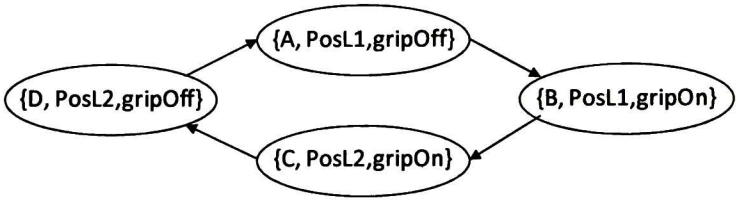


Figure 3.8. Checkpoint markings of CLSM in figure 3.7.

3.2.5. INTERRUPTIBLE T-COMPONENTS

Definition 3.6. Let X_e be a T-component of some plant component (G_e, M_e) in a CLSM IPN with a controller component (G_c, M_c), with (G_{c_k}, M_{c_k}) being one of simple cycles that compose it. If there exists $t_j, t_l \in T_e, j \neq l$ such that they have a permissive relationship with $p_i, p_r \in P_{c_k}, i \neq r$, respectively, then X_e is said to be *interruptible*. In words, a T-component is interruptible if at least two transitions in the component need permission from different places of the controller. In the CLSM of figure 3.7 the T-component (in the plant component 1) whose support is {gripOn, gripOff} is interruptible, because both transitions need a permission from different places of the controller.

Notice that if only one transition is enabled by a controller place, the t-component could be executed infinitely often while the controller place is marked; thus two transitions assure the controlled evolution of the t-component.

3.2.6. TRANSITION CAUSAL RELATIONS

For CLSM nets, it has been proved that if every t-component on the plant model is interruptible, then every cycle in the reachability graph of the system must contain at least one occurrence of a transition from the controller component [Lutz-Ley, 2011],[Lutz-Ley, 2012b]. This is possible only if the system reaches one checkpoint marking, thus ensuring that each cycle of states has at least one desired state.

An algorithm to decide this problem was presented in [Lutz-Ley, 2011],[Lutz-Ley, 2012b]. However, the algorithm proposed in these papers is computationally demanding since it requires the set of minimum t-components of the system. For this reason, an alternative method of calculation based on causal relations between transitions is presented below.

Definition 3.7. In an 1-bounded Petri net, a transition t_x ***depends*** on transition t_y if after one occurrence of t_x, one occurrence of t_y is needed in order for t_x to fire again. For example, in

figure 3.6, it can be seen that after the occurrence of moveToL1, the event *moveToCenter* must occur in order for moveToL1 to occur again, so it can be said that moveToL1 depends on *moveToCenter*.

Proposition 3.2. If a transition t_x depends on transition t_y , every transition included in any T-component also including t_x depends on t_y.

Proof: It is straightforward; it follows from the definition of a t-component.■

Proposition 3.3. If a transition t_x depends on transition t_y and t_z where t_y, t_z are transitions which require permission from different places on the specification, every t-component that includes t_x is ***interruptible***.

Proof: It follows straightforward from proposition 3.2.■

An example of proposition 3.3 can be seen on figure 3.7, specifically on plant component number 2, on the right part of the model. Transitions marked with ε depend on transitions *moveToCenter* and *moveToL2*, which require permissions from different places of the controller. Notice that every T-component including these ε transitions will be interruptible.

Proposition 3.4. Let T_{t_y} be the set of transitions which depend on transition t_y. $t_x \in T_{t_y}$ if :

1) $t_x = t_y$ or
2) $\exists p_a \in t_x\bullet$ such that $p_a\bullet \subset T_{t_y}$

Proof.

1) Straightforward
2) Once t_x fires, there is a token on p_a. Before t_x fires again, p_a must unmark as the net is 1-bounded (otherwise there would be two tokens on p_a). p_a only unmarks by the firing of some transition in T_{t_y}, as these fire again only after the firing of t_y (Definition 3.7) then p_a could unmark again only after the occurrence of t_y. It can be concluded that t_x depends on t_y ($t_x \in T_{t_y}$). ■

The above result and proposition 3.3 allows us to deduce an algorithm that determines whether all t components in the plant are interruptible. The pseudo code for the algorithm is presented in Algorithm 3.1.

Note that proposition 3.4 presents a sufficient condition only. In order to have a sufficient and necessary condition every plant component must be a strongly connected graph. It is easy to see that the execution complexity of the algorithm is bounded by $O(|T|^3)$ (the main body of the algorithm is a double nested loop that runs at most $|T|^3$ times).

Algorithm 3.1. determineInterruptible

Input: $\Sigma = (P, T, I, O)$: CLSM net where $P_c \subset P$ are the controller places
Output: *true or false.*

```
startset← ∅
for each p ∈ P_c{
        tag every transition t_p which requires permission from p with a label { p } (t_p is in the
plant model)
        startset←startset ∪ {t_p}
}
for each t ∈ startset{
        do{
                setChanged←false
                dependentSet← {t}
                for each e ∈ T{
                        if(∃p_a ∈ e• such that p_a• ⊂ dependentSet) then{
                                tag e with the same tag as t if it does
                                not have that tag already (e can have many tags);
                                dependentSet←dependentSet ∪ e
                                setChanged←true
                        }
                }
        }
}
if (Every transition on the plant model has at least two distinct tags) then return true
else return false
```

In this chapter, the state stability problem was analyzed for a class of Petri net models. More specifically, a technique to determine whether every cycle in a CLSM-modeled system includes at least one desired state has been developed. This technique is polynomial in time complexity. In the next chapter, the stability of a more general class of models, namely 1-bounded Petri nets, is studied.

Chapter 4 STABILITY OF 1-BOUNDED PETRI NETS USING UNFOLDINGS

A method for determining stability on a class of Petri nets was presented in the previous chapter. In order to find a method that is useful in a more general class of Discrete event systems, in this chapter the issue of stability of 1-bounded Petri nets is addressed by analyzing the Petri-net unfolding representation of the system.

4.1.PROBLEM STATEMENT

In chapter 3, the stability property has been defined on Petri nets. Also, a method to decide if every cycle of states in the system goes through a desired state has been presented. This approach works for a class called CLSM which is useful for the modeling and simulation of manufacturing processes and the algorithm has a polynomial-time complexity.

This chapter is devoted to address a more general problem: determining if an 1-bounded Petri net is stable with respect to some specified correct behavior. For this purpose, the Petri-net unfolding representation of the original net is going to be very useful.

Given an IPN model Σ_n describing the normal behaviour and an IPN model Σ_f including the normal and unwanted behaviour such that $\Sigma_n \subset \Sigma_f$, it must be determined if every cycle of the complete model includes at least one marking reachable on the normal behaviour submodel. This is stated below.

Let $\Sigma_f = (P, T, I, O, M_0) = (Q_f, M_0)$ be an 1-bounded Petri-net model where undesired transitions can occur, and $\Sigma_n = (P, T_n, I_n, O_n, M_0) = (Q_n, M_0)$ be a submodel of the same system that is free from undesired transitions, where $T_n \subset T$, $T_f = T - T_n$ where T_f are the undesired transitions, and $I_n(O_n)$ is the restriction of the input(output) function to transitions in T_n. The problem is *determining if every cycle of markings on* $R(Q_f, M_0)$ *includes at least one marking* $M \in R(Q_n, M_0)$. Notice that $R(Q_n, M_0) \subset R(Q_f, M_0)$ and $S = R(Q_n, M_0)$ includes all markings that are reachable without the firing of undesired events.

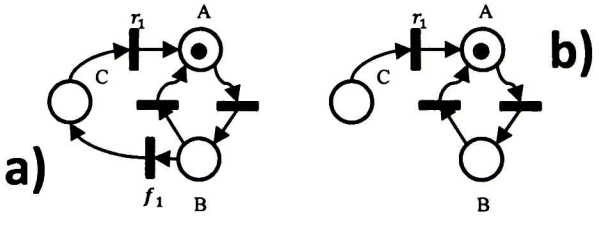


Figure 4.1. a) Complete behavior model, b) Desired behavior model.

This idea is illustrated in figure 4.1. In the model of figure 4.1.a there exists an undesired transition event f_1, which makes the system reach a marking {C}. In the model of figure 4.1.b the normal behavior submodel is represented and marking {C} is unreachable. It should be verified that every cycle of the complete model (4.1.a) goes through at least one state in the normal behaviour submodel (4.1.b) which is fullfiled in this example.

4.2.CYCLIC BEHAVIOR ON BRANCHING PROCESSES

In order to use branching processes to check for undesired state cycles, it is necessary to know how a cyclic behaviour can be detected in the branching process of a net. Since a 1-bounded net has a finite number of reachable markings, the unfolding of a cyclic 1-bounded net must contain events with repeated final markings. This idea is further developed next.

Proposition 4.1. Let (Q, M_0) be an 1-bounded net and $Unf = (O_Q, \rho)$ be its *unfolding* (infinite branching process). Then there exists a cycle in $R(Q, M_0)$ if and only if there are local configurations $[e_1], [e_2]$ in Unf with $[e_1] \subset [e_2]$ and $Mark([e_1]) = Mark([e_2])$.

Proof: ($\rightarrow$) Let $t_1 \dots t_n$ be a sequence such that $M \xrightarrow{t_1 \dots t_n} M$ Since $t_1 \dots t_n$ is a cyclic sequence, there is an infinite sequence of events $e_{x1} < e_{x2} < \cdots$ in Unf such that $\rho(e_{xk}) \in \{ t_1 \dots t_n\}$. As (Q, M_0) is 1-bounded, then $R(Q, M_0)$ is finite and $\exists e_1, e_2 \in \{e_{x1}, e_{x2}, \dots\}$ with $Mark([e_1]) = Mark([e_2])$ and $[e_1] \subset [e_2]$.

($\leftarrow$) If there are $[e_1], [e_2]$ in β with $[e_1] \subset [e_2]$ and $Mark([e_1]) = Mark([e_2])$ it is evident that there is a cycle starting and ending in $Mark([e_1])$ whose transitions are $[e_2] \backslash [e_1]$. ■

Notice that e_2 would be a cut-off event with respect to any adequate order that considers configuration size, so by Proposition 4.1 if there is a cycle on a system, a cut-off event is guaranteed to be reached when constructing a finite complete prefix.

Given that the unfolding of a net system with cyclic behavior is infinite, researchers have been using finite complete prefixes (FCP) of branching processes, as a valuable tool to deal with the complexity involved in the analysis of the reachability graph of a net system. Unlike the results presented in [McMillan 1993] and [Esparza 2002], the cutoff events are included (without their output conditions) as a part of the finite complete prefix to be analyzed. This is done because the presented properties are easier to evaluate if the cutoff events are added.

4.3.CYCLIC BEHAVIOR ON A FINITE COMPLETE PREFIX

Given that the finite complete prefix for a given unfolding Unf is just a finite initial part of Unf, it does not contain every event in Unf Then, a technique that allows detecting cyclic behavior on the FCP is needed. The following notions are useful to address this issue.

Definition 4.1. Let $Unf = (B', E', I', O', \rho')$ be *the unfolding* of an 1-bounded PN system. Unf contains a FCP $\beta = (B, E, I, O, \rho)$ obtained w.r.t. an adequate total ordering of the events. Let $e \in E$ be an event in β, the set of *overall causal precedence* of e, denoted as $e_{\ll}$, is defined as follows:

1) $e \in e_{\ll}$.

2) If $e_j, e_k \in E$ *and* $Mark([e_k]) = Mark([e_j]) \neq Mark([e])$, *with* $e_k \in e_{\ll}$ then $e_j \in e_{\ll}$.
3) If $e_j, e_k \in E$ *and* $(e_j < e_k)$ *where* $e_k \in e_{\ll}$, then $e_j \in e_{\ll}$.

In words $e_{\ll}$ is the set of events in E that have a directed path to the event e or they have a directed path to an event e_k such that e_k has the same final marking as another event which is already in $e_{\ll}$.

Remark 4.1. Notice that every event e_j in $e_{\ll}$ has a directed path starting from e_j to some event in Unf whose final marking is the same than that of e.

For example let e be the event 7 representing t_1 in figure 4.2. According to the definition, every event which has a path to event 7 is included in $e_{\ll}$, so events 4 and 1 are included. Also, every event in $e_{\ll}$ has a path to an event having the same local marking as event 7 (in this case, they have a path to event 7 itself).

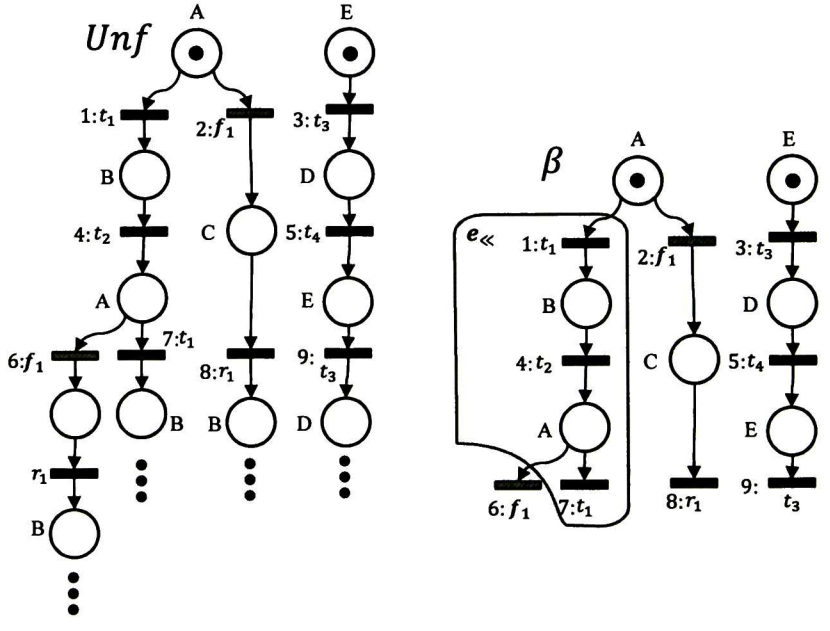


Figure 4.2. Overall causal precedence example.

Proposition 4.2 Let $Unf = (B', E', I', O', \rho')$ be *the unfolding* of an 1-bounded PN (Q, M_0). Unf contains a FCP $\beta = (B, E, I, O, \rho)$ w.r.t. an adequate total order. There is a cycle in (Q, M_0) if and only if there is a pair of events $e, e_j \in E$ such that e is a cutoff event, $e_j \in e_{\ll}$, and $Mark([e_j]) = Mark([e])$.

Proof. (→)Suppose that there is a cycle in (Q, M_0). By Prop 4.1 there are local configurations $[e_1], [e_2]$ in Unf with $[e_1] \subset [e_2]$ and $Mark([e_1]) = Mark([e_2])$. Suppose that $e_1 \in E$ and e_1 is not a cutoff event. If $e_2 \in E$, e_2 is a cutoff event ($C_1 \subset C_2$ implies $C_1 \prec C_2$) and $e_1 \in e_{2_{\ll}}$ (by the subsequent application of condition 3), then the property holds. If $e_2 \notin E$, $\exists e_3 \in E$ such that $[e_3] \subset [e_2]$ and e_3 is a cutoff event (e_3 being a cut off is the reason why $e_2 \notin E$) with $Mark[e_3'] = Mark[e_3]$ where e_3' is not a cutoff event, e_3' and e_3 have isomorphic finite

extensions, then $\exists e'_2 \in E'$ such that $Mark([e'_2]) = Mark([e_2]) = Mark([e_1])$ and $[e'_3] \subset [e'_2]$. Similarly to e_2, if $e'_2 \in E$ the condition holds ($e_1 \in e'_{2\ll}$ by subsequent application of 2 and 3, also e'_2 is a cut off event because it is supposed that e_1 is in E and the adequate order is total). Otherwise, $\exists e_4 \in E$ such that $[e_4] \subset [e'_2]$ and e_4 is a cutoff event and the same argumentation used for e_3 can be applied for e_4. Eventually, a cutoff event e_x is going to be found, such that $Mark([e_x]) = Mark([e_1])$ or $Mark([e_x]) = Mark([e_y])$ with $e_y < e_1$ (because the FCP was obtained w.r.t an adequate total order); then $e_1 \in e_{x\ll}$ or $e_y \in e_{x\ll}$ (by subsequent application of 2 and 3) so in any case there is a pair of events $e, e_j \in E$ such that e is a cutoff event. Then $e_j \in e_{\ll}$, and $Mark([e_j]) = Mark([e])$.

($\leftarrow$) Suppose that there is a pair of events $e, e_j \in E$ such that e is a cutoff event, $e_j \in e_{\ll}$, and $Mark([e_j]) = Mark([e])$ (β is obtained with respect with an adequate total order so e must be a cutoff event). If $e_j < e$ then $[e_j] \subset [e]$ and by Prop. 4.1 there is a cycle in (Q, M_0). If $e_j \not< e$, by the way $e_{\ll}$ is constructed, then there must exist pairs of events $(e_1, e'_1) \dots (e_m, e'_m)$ in β such that $Mark([e_h]) = Mark([e'_h])$ for all $h = 1 \dots m$, $e_j < e_m$, $e'_1 < e$ and $e'_i < e_{i-1}$ for all $i = 2..m$. Since $e'_1 < e$, it is true that $[e'_1] \subset [e]$. Also, $[e'_1]$ and $[e_1]$ have isomorphic finite extensions so $\exists e_{x1} \in B'$ (in Unf) such that $e'_2 < e_1 < e_{x1}$ and $Mark([e_{x1}]) = Mark([e])$ As $[e'_2]$ and $[e_2]$ also have isomorphic finite extensions the same argumentation can be applied. In this way, $\forall e_n \in e_1 \dots e_m$ there exists in Unf an event e_{xn} such that $e_n < e_{xn}$ and $Mark([e_{xn}]) = Mark([e])$. In particular, there exists an event e_{xm} with $e_j < e_m < e_{xm}$ so in Unf it holds that $[e_j] \subset [e_{xm}]$ and $Mark([e_j]) = Mark([e_{xm}])$. Then by Prop. 4.1 there is a cycle in (Q, M_0). ■

The above property can be used to determine if there is a cycle in (Q, M_0) only by analyzing the FCP (no reachability graph calculation is needed). In the next subsection, this result is used to detect undesired state cycles from the finite complete prefix of a Petri net system.

4.4. DECIDING STABILITY FROM THE FINITE COMPLETE PREFIX

The objective of this section is to explain a method that allows detecting the existence of undesired state cycles. First, the events have local configurations with desired final markings have to be found. Then, those events that have a desired final marking which are concurrent with another event whose final marking is undesired have to be identified. The first problem is handled next.

Definition 4.2. Let $\beta = (B, E, I, O, \rho)$ be a FCP branching process obtained w.r.t some adequate order. The set $normalNodes \subseteq B \cup E$ is defined as follows:

1) $Min(\beta) \in normalNodes$
2) If $e \in E$ with $\rho(e) \notin T_f$ and $\bullet e \subseteq normalNodes$ then $\{e \cup e\bullet\} \subseteq normalNodes$.
3) If $e, e_2 \in E$ with $Mark[e] = Mark[e_2]$ and $e_2 \in normalNodes$ then $\{e \cup e\bullet\} \subseteq normalNodes$.

Note that undesired events are not added directly when their inputs belong to $normalNodes$. The only case in which they could be added to $normalNodes$ is when there is already an event in $normalNodes$ which has the same final marking as some undesired event (by condition 3). Also, if there are events with undesired final markings they are not in $normalNodes$, this means that a configuration that includes events from $normalNodes$ only must have a desired final marking. The algorithm for finding the set $normalNodes$ can be easily deduced from the definition and is presented below.

Algorithm 4.1. findNormalNodes

Input: $\beta = (B, E, I, O, \rho)$: FCP BP of the system.
Output: *normalNodes*: Nodes that are reachable without the occurrence of undesired events.

```
normalNodes ← Min(β)
do{
        setChanged←false
        for each e ∈ E{
        if ((•e ⊂ normalNodes ∧ ρ(e) ∉ T_f) ∨ (Mark[e] = Mark[e2]|e2 ∈normalNodes))
        then {
                    normalNodes←normalNodes∪{e} ∪ e•
                      setChanged←true
              }
        }
}while(setChanged)
return normalNodes
```

In figure 4.3.a a 1-bounded Petri net is shown. The FCP obtained from 4.3.a is shown in 4.3.b and finally the normalNodes set is shown full colored in 4.3.c. Note that every event in *normalNodes* has a final marking which is reachable without the occurrence of the undesired event f_1.

Definition 4.3. Let $\beta = (B, E, I, O, \rho)$ be a FCP branching process obtained w.r.t some adequate order. The set $undesiredNodes \subseteq B \cup E$ is defined as follows:

$$undesiredNodes = \{y \in B \cup E \mid y \notin normalNodes\}$$

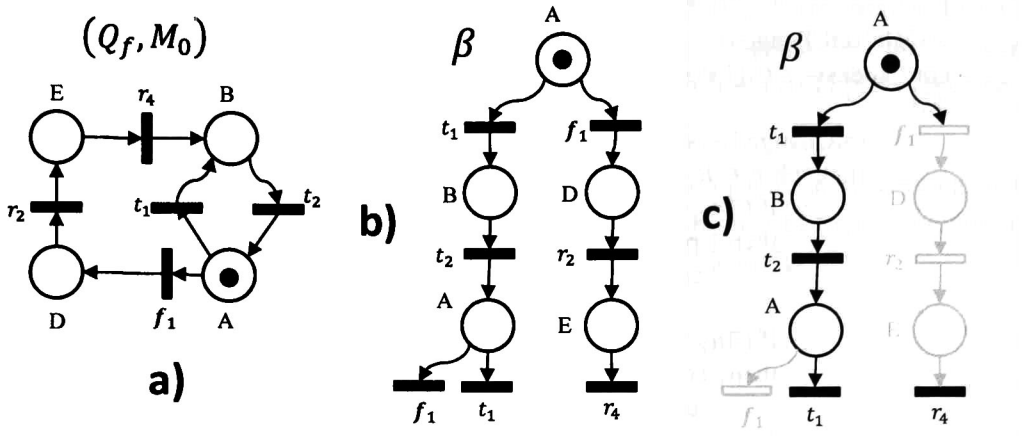


Figure 4.3. a) 1-bounded Petri net, b) a finite branching process representing its reachable markings. c) the normalNodes set.

If there are any events with undesired final markings, by definition they must be included in *undesiredNodes*. The next definition deals with identifying those events that can occur concurrently with an event having an undesired final marking.

Definition 4.4. Let $\beta = (B, E, I, O, \rho)$ be a FCP branching process obtained w.r.t some adequate order. The set $concurrentToUndesired \subseteq B \cup E$ is defined as follows:

$$concurrentToUndesired = \{x \in B \cup E \mid x \; co \; y, with \; y \in undesiredNodes\}$$

Note that every event whose occurrence ends in an undesired state in (Q, M_0) must either be in *undesiredNodes* set or in *concurrentToUndesired* set. The purpose of the next algorithm is to find the set *concurrentToUndesired*; first, for every node x in the *undesiredNodes* set, the nodes that are either causally related or conflicting with x are found. Since any pair of nodes are in a causal, conflict or concurrent relation, the concurrent related nodes are indirectly found.

Algorithm 4.2. findConcurrent:

Input: $\beta = (B, E, I, O, \rho)$: FCP of the system, $normalNodes$
Output: $concurrentToUndesired$ set.

tempConcurrent ←preNodes←posNodes← ∅
$undesiredNodes = (B \cup E) - normalNodes$
for each $x \in undesiredNodes$ {
    preNodes←posNodes← x. (nodes in causal relation w/ x)
    do{
        setChanged←false
        for each $n \in B \cup E$ {
            if ($\exists n_2 \in$ preNodes: $n_2 \in n\bullet$)
            then{preNodes←preNodes ∪ {n}
                setChanged←true
            }

```
        }
    }while(setChanged)
    confNodes← t ∈ (b•\preNodes): b ∈ B ∩ preNodes (nodes in conflict with x)
    do{
        setChanged←false
        for each n ∈ B ∪ E {
            if (∃n2 ∈ posNodes: n2 ∈ •n)
            then { posNodes←posNodes∪ {n}
                setChanged←true
            }
            if (∃n2 ∈ confNodes: n2 ∈ •n)
            then{ confNodes←confNodes∪ {n}
                setChanged←true
            }
        }
    }while(setChanged)
}
return (B ∪ E)\(preNodes ∪ posNodes ∪ confNodes)
```

An example of this set is shown in figure 4.4. First an 1-bounded net is displayed in figure 4.4.a; in figure 4.4.b, the finite complete prefix is shown, but only those nodes in the set undesiredNodes are shown with solid lines. Finally, in 4.4.c the nodes in the set concurrentToUndesired are shown with solid lines. Note that in the original net the transitions t_3,t_4 can occur while there is a token in place C. Accordingly, events labeled t_3,t_4 in β have a concurrent relation with the condition labeled C.

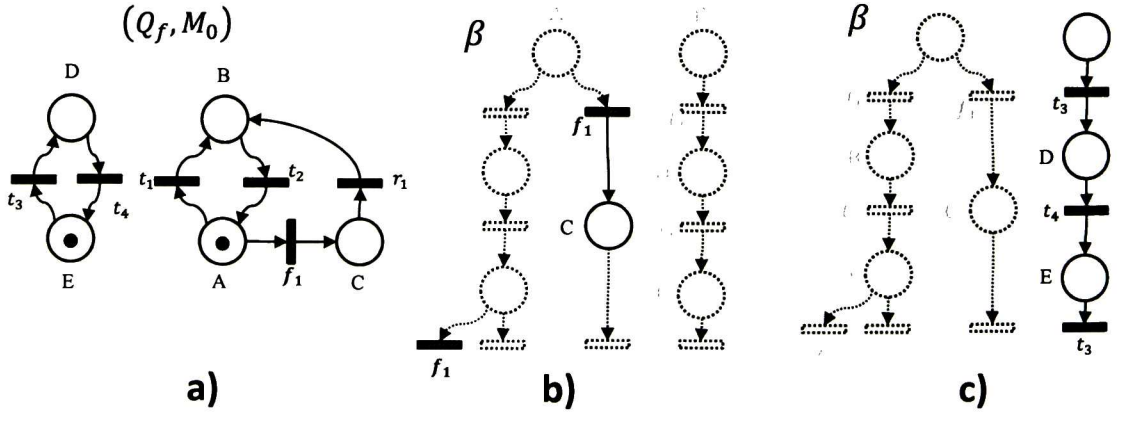


Figure 4.4. a) 1-bounded net, b) the undesiredNodes set and c) concurrentToUndesired set.

Definition 4.5. Let $\beta = (B, E, I, O, \rho)$ be a FCP branching process obtained w.r.t some adequate order. The graph $\beta^u = (B^u, E^u, I^u, O^u, \rho^u)$ is a subnet of β which is induced by the subset of

nodes ($undesiredNodes \cup concurrentToUndesired$) where I^u, O^u, ρ^u are similar to I, O, ρ but are restricted to B^u, E^u Also, given an event $e \in (undesiredNodes \cup concurrentToUndesired)$, $e^u_{\ll}$ is the overall causal precedence set of e restricted to nodes in β^u

If there is an event representing a transition that could occur in an undesired marking in the original net whose firing also ends in an undesired marking, then it must be in β^u The nodes in the set β^u for the net shown in figure 4.4. are depicted with solid lines in figure 4.5.

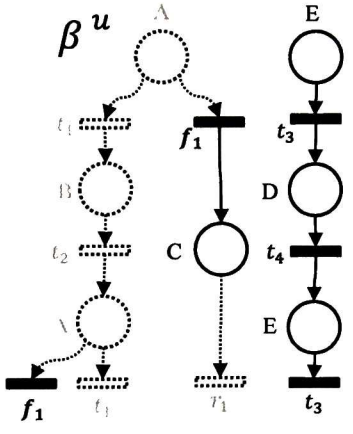


Figure 4.5. β^u for the PN in figure 4.4.

Proposition 4.3 Let $Unf = (B', E', I', O', \rho')$ be *the unfolding* of an 1-bounded PN (Q, M_0). Unf contains a FCP $\beta = (B, E, I, O, \rho)$ w.r.t an adequate total order. Also $\beta^u = (B^u, E^u, I^u, O^u, \rho^u)$ is a subnet of β obtained as described in definition 4.5. A necessary condition for the existence of a undesired marking cycle in (Q, M_0) is that there exist a pair of events $e, e_j \in E^u$ such that e is a cutoff event, $e_j \in e^u_{\ll}$, and $Mark([e_j]) = Mark([e])$.

Proof. Suppose that there is a cycle of undesired markings in (Q, M_0) but there is no pair of events $e, e_j \in E^u$ such that e is a cutoff event, $e_j \in e^u_{\ll}$, and $Mark([e_j]) = Mark([e])$. Since (Q, M_0) is cyclic, there is a cutoff event $e \in E$ such that $\exists e_2 \in e_{\ll}$ with $Mark[e] = Mark[e_2]$ (Proposition 4.2). Also, both $\rho(e)$ and $\rho(e_2)$ must be able to occur while (Q, M_0) is in an undesired marking; this is only possible if these events are in the *undesiredNodes* or *concurrentToUndesired* subsets, thus they can be found in β^u. However $e_2 \notin e^u_{\ll}$ (by supposition); so there exists $e_3 \in e_{\ll}$ such that $e_3 \notin E^u$ and also it prevents e_2 from being in $e^u_{\ll}$. Then there must exist events $e'_3, e' \in B'$ (in Unf) with $Mark[e_3] = Mark[e'_3], Mark[e] = Mark[e']$ and $e_2 < e'_3 < e'$ such that e'_3 is in *normalNodes* and is not concurrent to any node in *undesiredNodes*. This means that $\rho(e'_3)$ is not able to occur from an undesired marking, so there is no cycle of undesired markings and a contradiction is obtained. ■

Based on the previous result, a strategy for deciding if a 1-bounded net is free from undesired state cycles is obtained and is described next. First, β^u is obtained. After that, for every cutoff event e in β^u, $e^u_{\ll}$ is obtained. If for every event e in β^u, $e^u_{\ll}$ does not contain an event e_j with $Mark[e] = Mark[e_j]$ then the original net is free from undesired state cycles. This procedure is detailed in pseudo code in the following algorithm.

Algorithm 4.3. checkNoCycle:

Input: $\beta^u = (B^u, E^u, I^u, O^u, \rho^u)$: FCP of the system.
Output: *true or false*.

```
for each e ∈ E^u such that e is cutoff{
    causalSet ← {e};
    for each n ∈ (B^u ∪ E^u){
        if ((∃n_2 ∈ causalSet: n_2 ∈ n•) ∨
        (n ∈ E^u ∧ Mark[n] = Mark[n_2], n_2 ∈ causalSet, n_2 ≠ e))
        then{
                causalSet←causalSet∪ {n}
                if(n ∈ E ∧ Mark[n] = Mark[e])
                then{
                    return false
                }
        }
    }
}
return true
```

4.5.EXAMPLE

Simple process communication

In the following example, it is verified that the system has no undesired-state cycles using the results previously obtained. As depicted in figure 4.6, there are two simple processes that communicate by messages. Each process can have an undesired occurrence; the undesired events are labeled as f_1 and f_2.

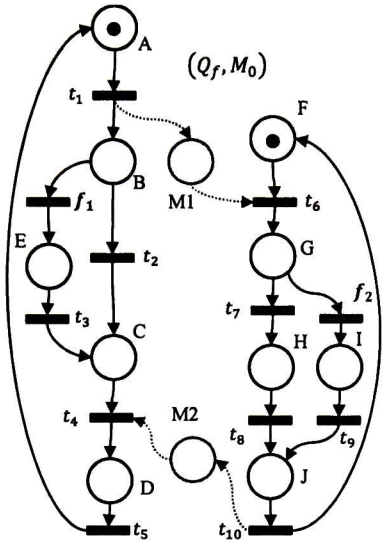


Figure 4.6. Two Simple communicating processes.

Making use of the adequate total order presented by (Esparza et. al. 2002) the finite complete prefix $\beta = (B, E, I, O, \rho)$ obtained with respect to such order is depicted in figure 4.7. Every event is tagged using its identifier on β first, followed by ":" and the transition identifier that it represents. The cut off events are 3,9 and 11.

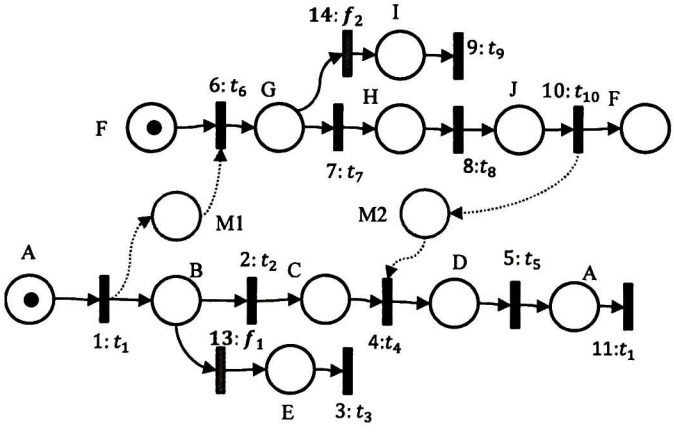


Figure 4.7. Finite complete prefix of system in figure 4.6.

First, β has to be analyzed to determine which events have configurations with desired final markings (markings in $R(Q_n, M_0)$). This can be done by using the algorithm 4.1 which computes the *normalNodes* set. The result of this analysis is graphically shown in figure 4.8.

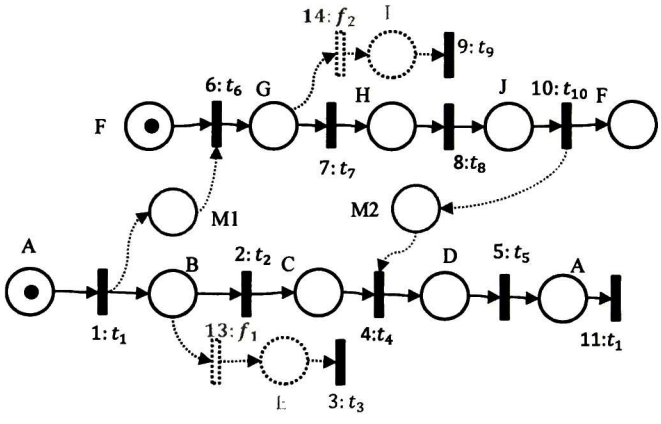


Figure 4.8.Transitions having desired final markings are shown with solid lines.

Once the undesiredNodes set is obtained (shown with dashed lines in figure 4.8), the next step is to obtain the nodes which are concurrent to undesiredNodes. This is done by algorithm 4.2 and the resulting set, *concurrentToUndesired* is depicted in solid lines in figure 4.9. As the *undesiredNodes* set is already included in the *concurrentToUndesired* set, figure 4.9 also shows β^u in solid lines.

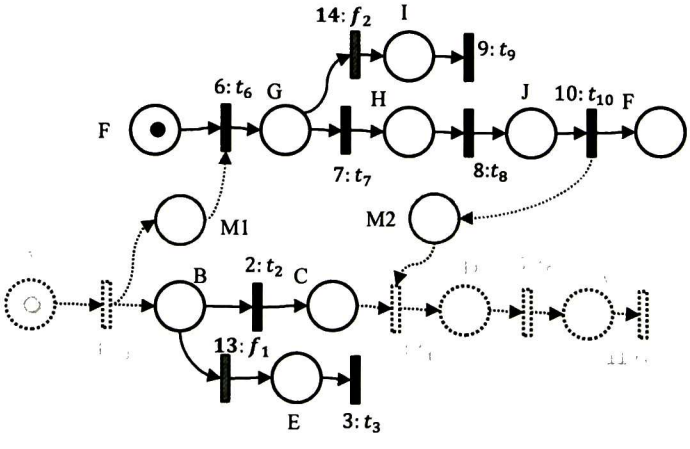


Figure 4.9. *concurrentToUndesired* set is shown (in this case β^u is also shown).

Finally, for every cutoff event e in β^u, it should be checked that $\nexists e_j \in e^u_{\ll}$ such that $Mark[e] = Mark[e_j]$. In figure 4.10, the two cutoff events 9 (4.10.a) and 3 (4.10.b) are shown in solid lines with their respective overall causal sets. As can be verified on the figure, there is no event e_j in the overall causal set of event 9 (event 3) such that e_j has the same final marking than event 9

(event 3). For this example it can be concluded that the original net is free of undesired state cycles.

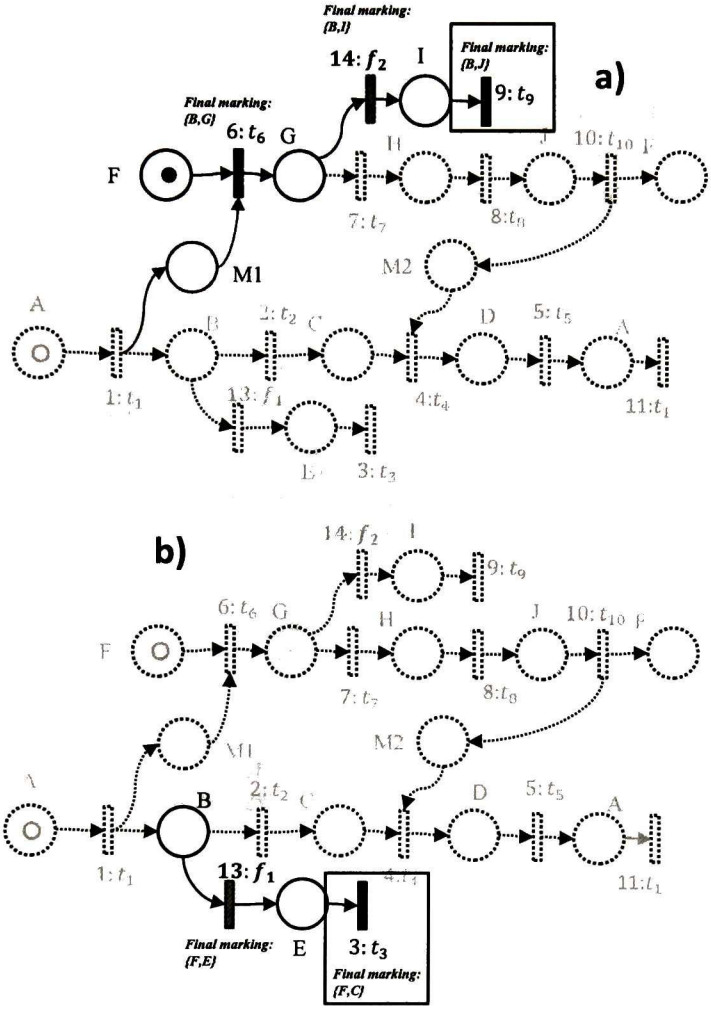


Figure 4.10. The FCP is undesired-state cycle free. $9_{\ll}$ and $3_{\ll}$ are shown in solid lines in a) and b) respectively.

4.6. K-STABILITY ANALYSIS

Once it is known that a system is returning to a desired state in a finite number of steps, it is useful to determine the maximum number of steps required to return to a desired state; this is especially important in critical fault-tolerant systems, where a fast return to the desired behaviour is required. In this sense, the property of k-stability is very useful. Given a complete 1-bounded IPN model Σ_f (free from undesired marking cycles) including the normal and unwanted

behaviours and a IPN submodel Σ_n describing the normal behaviour only, such that $\Sigma_n \subset \Sigma_f$, it should be verified that starting from any marking of the complete model, at least one marking reachable on the normal behavior submodel (a desired state) is reached within at most k steps. This is stated below.

4.6.1. THE K-STABILITY PROBLEM

Let $\Sigma_f = (P, T, I, O, M_0) = (Q_f, M_0)$ be a 1-bounded Petri-net model where undesired transitions can occur and it is free from undesired marking cycles, and $\Sigma_n = (P, T_n, I_n, O_n, M_0) = (Q_n, M_0)$ be a submodel of the same system which does not include undesired transitions, where $T_n \subset T$, $T - T_n = T_f$ where T_f are the undesired transitions, and $I_n(O_n)$ is the restriction of the input(output) function to transitions in T_n. The problem is *determining if starting from any marking on* $R(Q_f, M_0)$ *the system is guaranteed to reach a marking on* $M \in R(Q_n, M_0)$ *within at most k steps.*

For example, for the Petri net model of figure 4.6, when the net reaches an undesired marking by the firing of either f_1 or f_2, the maximum number of firings that can occur before the system reaches a desired state is 4 (after f_1: $t_6 t_7 t_8 t_{10}$, in case that the net reaches an undesired marking by the firing of f_2, the number of firings is smaller). Then it can be said that the Petri net system of figure 4.6 is *5-stable*.

4.6.2. FINDING THE LONGEST PATH TO THE DESIRED STATES.

In order to find the value of k such that a Petri net system is k-stable, the longest event sequence that reaches undesired markings only has to be found. Furthermore, this has to be done by analysing the finite complete prefix instead of the unfolding. Fortunately, the definitions presented in the above paragraphs are useful for this task; as it has been noted, if the system reaches an undesired marking after the firing of some transition t, then there must exist an event in β^u that has the label t. This implies that the size of E^u (events of β^u) is an upper bound of the number k (system is $k - stable$ for $k \geq |E^u|$). The next concepts are useful for finding a bound that is closer to the value of k.

Definition 4.6. Let $Unf = (B', E', I', O', \rho')$ be *the unfolding* of an 1-bounded PN system. Unf contains a FCP $\beta = (B, E, I, O, \rho)$ obtained w.r.t. an adequate total ordering of the events. Also $\beta^u = (B^u, E^u, I^u, O^u, \rho^u)$ is a subnet of β obtained as described in definition 4.5 such that $\nexists e, e_j \in E^u$ with e being a cutoff event, $e_j \in e^u_{\ll}$, and $Mark([e_j]) = Mark([e])$. Let $n \in (B^u \cup E^u)$ be a node in β^u. *The maximal event set following n in* β^u, denoted as $n^u_{\gg}$, is defined as follows:

1) If $n \in B$ then $n^u_{\gg} = {e_1}^u_{\gg}$ where $e_1 \in n\bullet$ and $|{e_1}^u_{\gg}| \geq |{e_x}^u_{\gg}|, \forall e_x \in n\bullet$ (If n is a place, its maximal event set following n is $e_{1\gg}$, where e_1 is the successor of n with the biggest *maximal event set following* the event).
2) If $n \in E$ is not a cut off event then $n^u_{\gg} = (\cup_{i=0}^{m} b^u_{i\gg}) \cup \{n\}$ where $\{b_0 \dots b_m\} = n\bullet$ (If n is an event that is not a cut off, its *maximal event set following n* is the union of those *maximal event sets following* its succesors plus n itself)

3) If $n \in E$ is a cut off event with $Mark([n]) = Mark([e])$ where e is not a cut off event, then $n_{\gg}^{u} = (\bigcup_{i=0}^{m} b_{i\gg}^{u}) \cup \{n\}$ where $\{b_0 \dots b_m\} = e\bullet$ (If n is a cut off event, its *maximal event set following* n is equal to the union of the *maximal event sets following* the successors of the event e which has the same final marking of n and it is not a cut off event) .

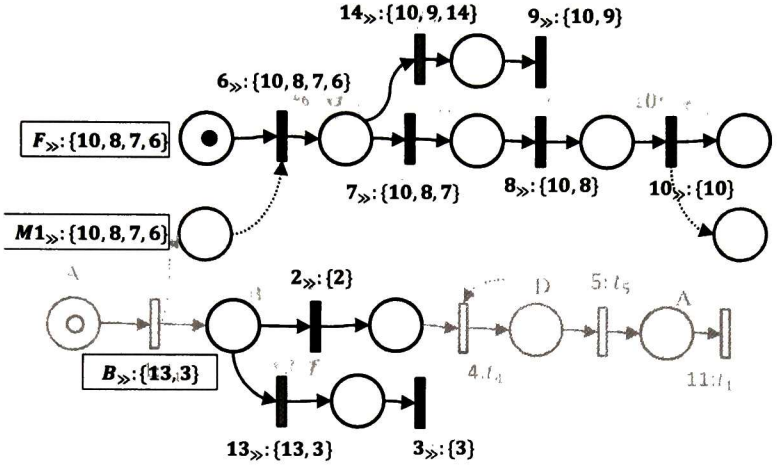


Figure 4.11. Computation of $B_{\gg}$, $M1_{\gg}$ and $F_{\gg}$.

In words, $n_{\gg}^{u}$ includes the longest possible sequence of events that can occur after n in β^{u}. For example, the maximal sequence set was calculated for the nodes B,$M1$ and F from the β^{u} graph shown in figure 4.9 and the results are shown in figure 4.11; For the place B, $B_{\gg}$ is defined as being equal to $13_{\gg}$ because $13_{\gg}$ is larger than $2_{\gg}$.Note that the restriction of that $\nexists e, e_j \in E^{u}$ such that e is a cutoff event, $e_j \in e_{\ll}^{u}$, and $Mark([e_j]) = Mark([e])$ (the system does not have the necessary condition for the existence of undesired cycles) ensures that $n_{\gg}^{u}$ is finite for $n \in (B^{u} \cup E^{u})$. This claim is further formalised in the next proposition.

Proposition 4.4. Let $Unf = (B', E', I', O', \rho')$ be *the unfolding* of an 1-bounded PN (Q, M_0). Unf contains a FCP $\beta = (B, E, I, O, \rho)$ w.r.t an adequate total order. Also $\beta^{u} = (B^{u}, E^{u}, I^{u}, O^{u}, \rho^{u})$ is a subnet of β obtained as described in definition 4.5 such that $\nexists e, e_j \in E^{u}$ with e being a cutoff event, $e_j \in e_{\ll}^{u}$, and $Mark([e_j]) = Mark([e])$. Then $n_{\gg}^{u}$ is defined and finite for all nodes n in β^{u}.

Proof. Suppose that $\nexists e, e_j \in E^{u}$ such that e is a cutoff event, $e_j \in e_{\ll}^{u}$, and $Mark([e_j]) = Mark([e])$.

In particular, it should be verified that $e_{\gg}^{u}$ is defined and finite for all cut off events e in β^{u} because if this is true then it is also true for every other node. Also, this is true only if for every cut off event e with $Mark(e) = Mark(e_x)$,where e_x is not a cut off event, $e \notin e_{x\gg}^{u}$. Suppose that e_1 is a cutoff marking and also suppose that $Mark(e_1) = Mark(e_2)$ where e_2 is not a cut off

event. It is evident that $e_2 \nprec e_1$ (by supposition $e_2 \notin e^u_{1\ll}$) also there is no set of events $\{e_3, e'_3, e_4, e'_4 \dots e_j, e'_j\}$ so that $e_2 < e_3 \quad e'_j < e_1 \quad e_h$ is a cut off event with $Mark(e_h) = Mark(e'_h)$ where e'_h is not a cut off and $e'_h < e_{h+1}$ for $h = \{3 \dots j-1\}$ (if this were true, then $e_2 \in e^u_{1\ll}$). Then there is no way that e_1 can be added to $e^u_{2\gg}$. Then $e^u_{2\gg}$ is defined and finite and does not contain e_1 (for any e_1, e_2), this means that $e^u_{\gg}$ is defined and finite for all cut off events e, finally $n^u_{\gg}$ is defined and finite for all nodes n in β^u. ■

When n is a cut off event, $n^u_{\gg} = (\bigcup_{i=0}^{m} b^u_{i\gg}) \cup \{n\}$ where $\{b_0 \dots b_m\} = e\bullet$ with $Mark([n]) = Mark([e])$, this is because e and n have isomorphic finite extensions, then whichever event that could be fired after the occurrence of e must also be possible to fire after the occurrence of n, however this information is not present in the FCP (n is a cut off event). For example, in figure 4.11, although the event 9 is a cutoff event, $9_{\gg}$ includes the event 10 because after the firing of 9, an occurrence of event 10 is possible in the original Petri net (refer to figure 4.6).

Definition 4.7. Let $Unf = (B', E', I', O', \rho')$ be *the unfolding* of an 1-bounded PN system. Unf contains a FCP $\beta = (B, E, I, O, \rho)$ obtained w.r.t. an adequate total ordering of the events. Also $\beta^u = (B^u, E^u, I^u, O^u, \rho^u)$ is a subnet of β obtained as described in definition 4.5 such that $\nexists e, e_j \in E^u$ with e being a cutoff event, $e_j \in e^u_{\ll}$, and $Mark([e_j]) = Mark([e])$. The *Maximal Sequence Set* represented by MSS is defined as $MSS = \bigcup_{i=0}^{r} n^u_{i\gg}$ where $\{n_0 \dots n_r\} \subset B^u \cup E^u$ are the nodes in β^u with an empty preset.

For example, the MSS that is obtained from the β^u graph of figure 4.9 is the union of $B_{\gg}$, $M1_{\gg}$ and $F_{\gg}$. These sets are shown enhaced in figure 4.11. Then $MSS = \{13,10,8,7,6,3\}$.

Proposition 4.5. Let $Unf = (B', E', I', O', \rho')$ be *the unfolding* of an 1-bounded PN (Q, M_0). Unf contains a FCP $\beta = (B, E, I, O, \rho)$ w.r.t an adequate total order. Also $\beta^u = (B^u, E^u, I^u, O^u, \rho^u)$ is a subnet of β obtained as described in definition 4.5 and MSS is the Maximal Sequence Set as described in definition 4.7. Also, suppose that $\nexists e, e_j \in E^u$ such that e is a cutoff event, $e_j \in e^u_{\ll}$, and $Mark([e_j]) = Mark([e])$. Then (Q, M_0) is $k-$stable for $k \geq |MSS|$.

Proof. Let $L = |MSS|$. Suppose that (Q, M_0) is not $k-$stable for $k \geq |MSS|$. This means that there exists some event firing sequence $\sigma = t_1 t_2 \dots t_L$ of length L such that $m_j \xrightarrow{t_1} m_{j+1} \xrightarrow{t_2} m_{j+2} \xrightarrow{t_3} \dots \xrightarrow{t_L} m_{j+L}$ where m_{j+i} is an undesired marking for $i \in \{0, \dots L\}$ (there are $L+1$ undesired markings). Then, there are configurations $C'_j, C'_{j+1}, C'_{j+2}, \dots C'_{j+L}$ in Unf (infinite unfolding) such that $Mark[C'_{j+i}] = m_{j+i}$ and $C'_{j+v+1} = C'_{j+v} \cup e'_{v+1}$ where the label of e'_{v+1} is t_{v+1} for for $i \in \{0, \dots L\}, v \in \{0, \dots L-1\}$. Also, as β was obtained w.r.t. an adequate order, it is complete and it contains configurations $C_j, C_{j+1}, C_{j+2}, \dots C_{j+L}$ such that $Mark[C_{j+i}] = m_{j+i}$ and C_{j+v} can be extended with event e_{v+1} for $i \in \{0, \dots L\}$, $v \in \{0, \dots L-1\}$. Note that for all e_{v+1} with

$v \in \{0, \ldots L-1\}$ their firing results in an undesired marking so they are in β^u Also, as configuration C_j has an undesired final marking it must contain an event e_0 from β^u

If β^u is conflict free, then all of the events of β^u are in MSS (events are only discarded in conflict places when constructing MSS) but $|\{e_v| \ v \in 0, \ldots L\}| = L + 1$ so β^u has to have conflict, otherwise a contradiction is found (all of the events e_i are in β^u and all of the events in β^u are in MSS so the number of events e_i should be less or equal than L).

As $|\{e_v| \ v \in 0, \ldots L\}| = L + 1$ is greater than $|MSS|$, at least one event in $\{e_v| \ v \in 0, \ldots L\}$ is not in MSS. Let e_q be such an event. It is possible that $e_q \in e_{x\gg}$ with $|e_{y\gg}| \geq |e_{x\gg}|$ and so for the predecessor conflict place $n1$, $n1_{\gg} = e_{y\gg}$ (from Definition 4.6 number 1) and then e_q is not in MSS but also $|e_{y\gg}| \geq |e_{x\gg}|$ and in this case it should be true that $L \geq |\{e_i| \ i \in 0, \ldots L\}|$ which is a contradiction.

Finally, note that in β it is true that C_{j+v} can be extended with event e_{v+1} for $v \in \{0, \ldots L-1\}$ but this does not necessarily mean that if $C_{j+v+1} = C_{j+v} \cup e_{v+1}$ then $C_{j+v+2} = C_{j+v} \cup e_{v+1} \cup e_{v+2}$ (as was true in Unf for C'_{j+v+1} and C'_{j+v+2}) because e_{v+1} could be a cut off event. However this has already been considered in (Definition 4.6 number 3) so for a cut off event e, $e_{\gg}$ has at least the same number of elements it would have had if e was not a cut off event. So the contradiction found in the above paragraph still holds for branches in conflict that include cut off events. Then, (Q, M_0) is k −stable for $k \geq |MSS|$.■

The last proposition is useful for finding an upper bound on the value of K; (Q, M_0) is k −stable for $k \geq |MSS|$. For example, in the case of the β^u graph shown in figure 4.9 and 4.11, $MSS = \{13,10,8,7,6,3\}$, $|MSS| = 6$ and then the system should return to a desired state in at most 6 steps. Since it has been determined that, for the analyzed system, the maximum number of event occurrences that end in an undesired marking is 4, the obtained result holds true.

In this chapter, the stability of DESs modelled with Petri nets is studied avoiding the reachability analysis. The property has been defined and characterized for systems expressed by 1-bounded IPN. The proposed sufficient condition for the stability of 1-bounded nets provides a more efficient procedure by making use of the branching processes; this algorithm has polynomial-time complexity on the number of nodes of the branching process. Also, a method to determine a bound on the value of k such that a Petri net system is k-stable was presented.

Chapter 5 IMPLEMENTATION AND TESTING

This chapter presents the application of the method described in chapter 4. First, the algorithm presented in chapter 4 was implemented as a software tool that is able to verify a Petri net in the PNML format. Also, the Petri net model for the two-phase commit protocol was obtained. The model is easily scalable by including more processes. Finally, the resulting Petri net was analyzed by the implemented method.

The algorithms for verifying stability from the reachability graph and from the Petri net unfolding were implemented as a software tool. In this chapter, some of the results of the use of these methods and a comparison of their execution times are presented.

5.1. SOFTWARE IMPLEMENTATION OF THE STABILITY ANALYSIS ALGORITHMS

The main interface of the software is depicted in figure 5.1. In order to be able to analyze the stability of a Petri net, the user has to open a PNML (Petri Net Markup Language)[Hillah 2009] file first using the *Select* button. This prompts the software to show an open file dialog. When the user correctly selects a PNML file, the information of the selected PNML is shown in the text box of the main window. Also, the buttons *Verify Stability using Reachability Graph* and *Verify Stability using Petri net Unfolding* which were initially disabled are enabled at this point (after selecting a PNML file), allowing the user to choose an analysis method and start the verification process.

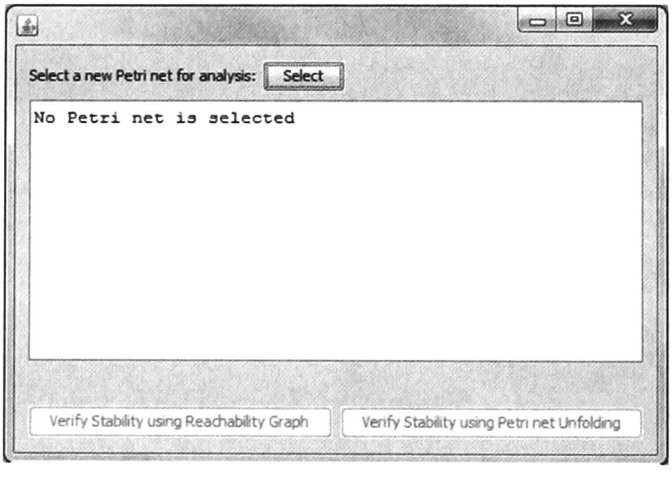


Figure 5.1. The main interface of the software tool.

An overview of the architecture of the software tool is shown in figure 5.2. Once the user has selected a PNML file, the method createFromFile creates a PetriNet Object from the PNML file. This is the PetriNet object used for the analysis. When the user clicks on *Verify Stability using Reachability Graph*, the PetriNet object is passed as an argument to construct a ReachabilityGraph object, which is passed as an argument to the method StabilityChecker.isStable(ReachabilityGraph object). This method returns true when the analyzed Petri net is stable and false otherwise. If the user clicks the *Verify Stability using Unfolding*

button, then the PetriNet object is passed as an argument to construct a BranchingProcess object which is passed as an argument to the method StabilityChecker.isStable(BranchingProcess object) (the method is overloaded). This method returns true if the analyzed Petri net is stable, and false if the sufficient condition for stability presented in chapter 4 does not hold for the analyzed Petri net.

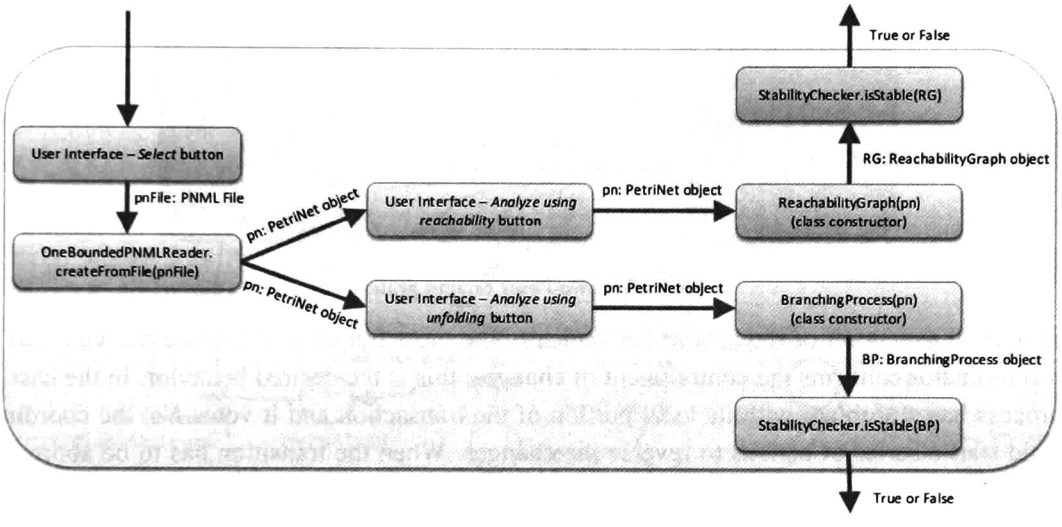


Figure 5.2. System's architecture overview.

5.2. CASE STUDY: THE 2-PHASE COMMIT PROTOCOL

A transaction is a set of operations that by definition has the following properties: Atomicity, Consistency, Isolation and Durability. A distributed transaction is a transaction in which the set of operations is to be executed by two or more hosts. Like any other transaction, a distributed transaction must have the four properties mentioned above. A common algorithm for the correct completion of a distributed transaction is the two-phase commit protocol.

The basic algorithm for the two-phase commit protocol is shown in figure 5.3. It is assumed that one of the hosts has the role of a *Coordinator*. The algorithm starts when the coordinator sends a query to all other processes involved in a transaction to solve if they are going to commit the changes made. After the processes answer this query, the coordinator confirms the commitment to all other processes only if all processes voted yes, and instructs a full transaction rollback otherwise. Finally, the Coordinator waits for the confirmation of the completion of the requested actions.

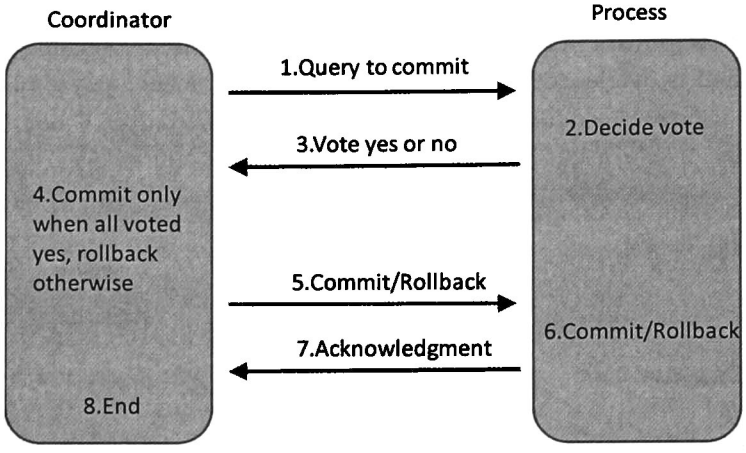


Figure 5.3. Two-Phase commit protocol.

The normal execution of the protocol is typically the most frequent; All processes vote *Yes* and the coordinator confirms the commitment of changes; this is the desired behavior. In the case that a process has a problem with its local portion of the transaction and it votes *No*, the coordinator should start a series of actions to reverse the changes. When the transition has to be aborted and the rollback of changes is complete, the system should reach a global state that allows resuming the normal behavior in a finite number of steps.

5.2.1. PETRI NET MODEL FOR A TWO-PHASE COMMIT IMPLEMENTATION

The Petri net model for an implementation of the Two-phase commit protocol is shown in figures 5.4 and 5.5. These are the models for a Coordinator and a single process respectively and are shown in different images for reasons of clearness and size. The model is scalable; there is only one coordinator but several processes can be added. The composition of the coordinator with the processes is performed through places that act as message mailboxes.

The coordinator starts in place *idleCoordinator* where the transition *queryToCommit* can be fired. This puts a token in each of the *queryToCommitMSG(i)* places, one for each process *i*. After this is done, the coordinator waits to receive the votes of the processes. The transition *commitReached* is enabled only when all processes vote to commit the transaction. After *commitReached* occurs, the transition *sendCommitAlert* puts one token in each of the *commitAlert(i)* places. When all processes have sent their acknowledgement, the transition *complete* is enabled and after its firing the coordinator returns to its starting place. If some process *k* votes to abort the transaction, the transition *pkAbort* is fired. Then *sendRollbackAlert* puts one token in each of the *rollbackAlert(i)* places. After receiving the acknowledgement, the coordinator enters the clearChannels place, where the tokens (messages) that were not consumed (as a result of the coordinator only consuming the abort token of process *k*) are consumed by transitions clearVoteA(i) and clearVoteC(i), to clear either a vote abort or a vote commit generated by process *i*. This serves the

purpose of keeping the net 1-bounded. After all messages are cleared, the transition *clearOk* is activated, enabling the coordinator to return to its starting place.

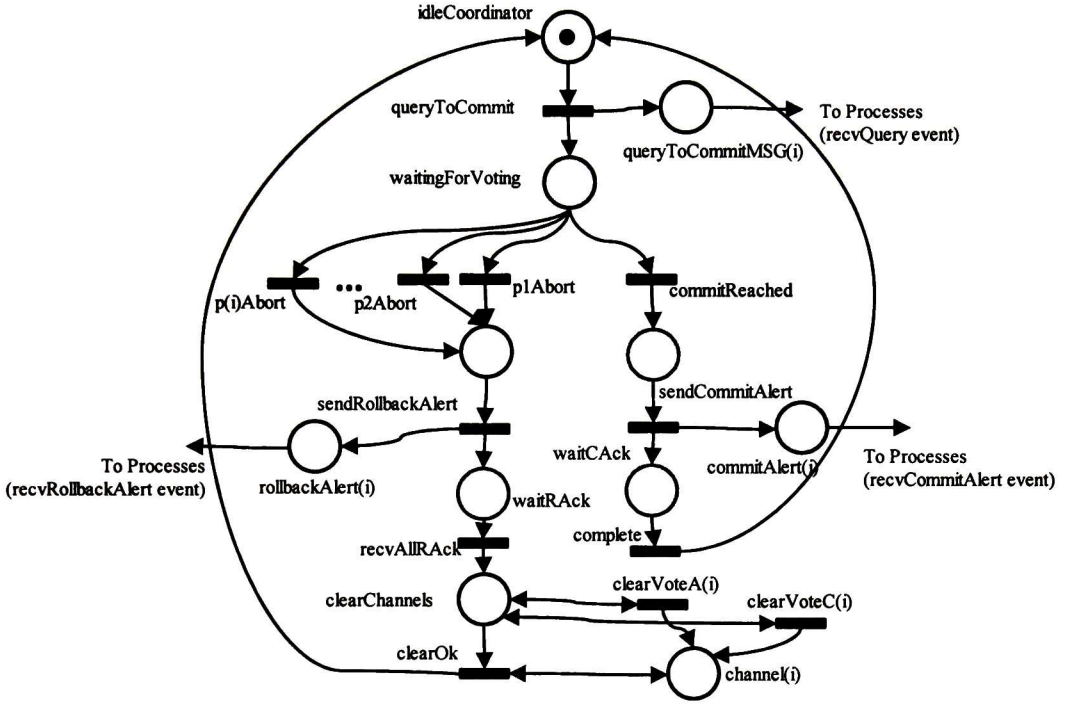


Figure 5.4. Coordinator model of the two phase commit protocol.

The process *i* starts in the place *idleP(i)*. The event *recvQuery* is enabled after the Coordinator fires the *queryToCommit* event. When the *decide* place is marked, the process has the choice to fire either the *voteAbort* or the *voteCommit* event; this sends the corresponding message to the Coordinator. In particular, the *voteAbort* event is modeled as an abnormal event. After the vote has been generated, the process waits until the coordinator makes the final decision and sends a *rollbackAlert* or a *commitAlert*. In either case, the process sends an acknowledgement to the Coordinator and returns to its initial place.

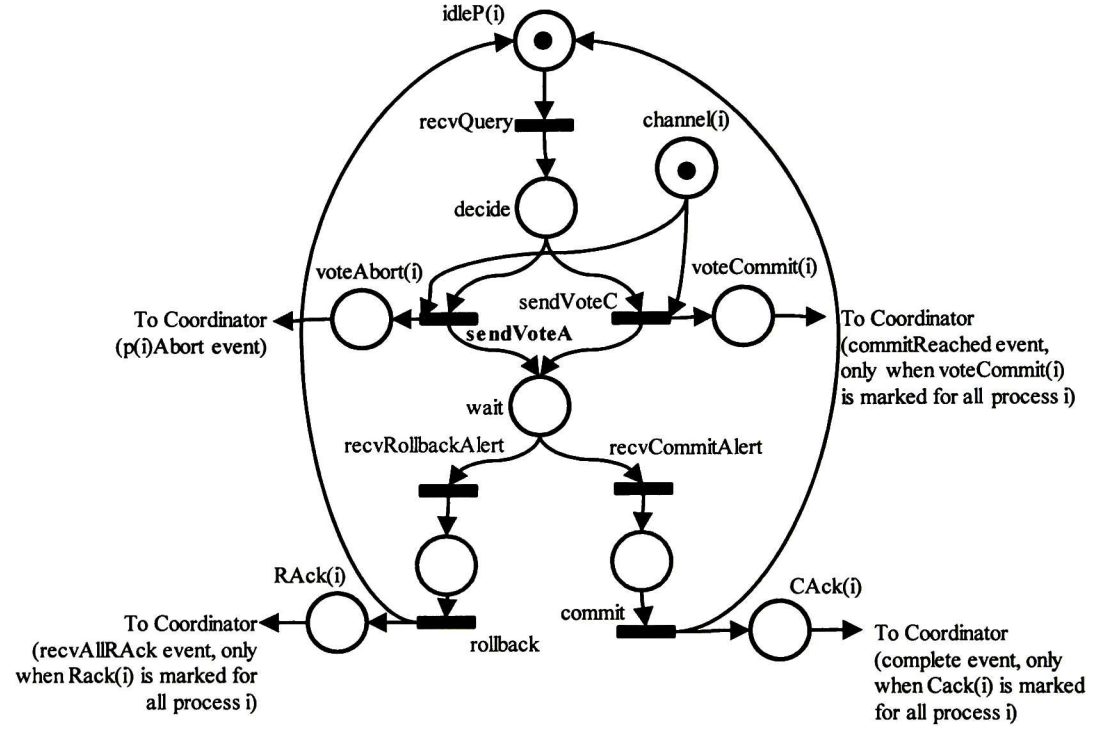


Figure 5.5. Process of the 2-phase commit protocol.

5.3. TESTING THE TOOL THROUGH THE 2-PHASE COMMIT MODEL

The stability of the 2-phase commit Petri net model was verified using the software tool presented in section 5.1. The stability of the Petri net was verified using both methods (using the reachability graph and using the method presented in chapter 4) for systems of 2 to 8 processes. The testing environment is described below:

Hardware Environment:

Intel Core 2 Duo CPU T6500 at 2.10 GHz.
RAM: 4GB.

Software Environment:

Windows 7 Home Premium 64 bit.
Java SDK 6 Update 20.
Java 6 Update 20.
NetBeans 7.2.
PIPE [Dingle 2009] (for the creation of the PNML files)

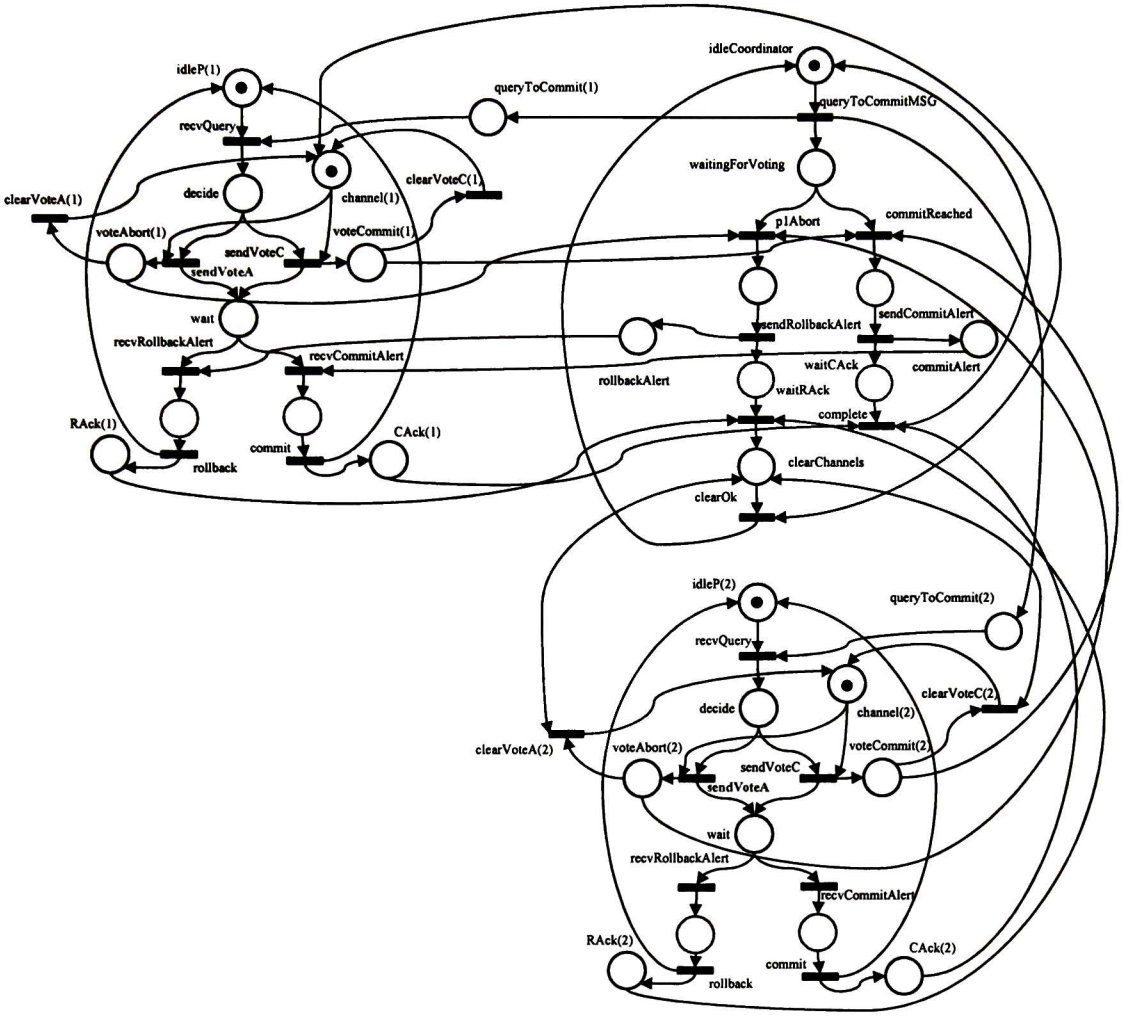


Figure 5.6. Two-phase commit protocol example with one coordinator and two processes.

All tests were made running the source code within NetBeans. The performance of the tests results are shown in figure 5.7.

Note that the vertical axis (execution time) has a logarithmic scale. With the exception of the results for a model with 2 processes, the algorithm proposed in this document has a shorter running time than obtaining and using the reachability graph to employ algorithms for finite automata stability.

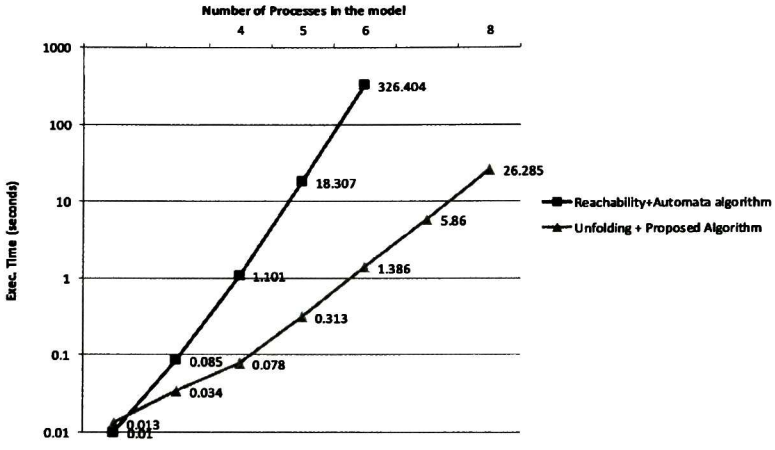


Figure 5.7. Graph showing the differences in execution time between both algorithms. The horizontal axis represents the number of processes in the model while the vertical axis is the execution time in seconds.

Moreover, the advantage in time efficiency appears to grow as the number of processes in the Petri net model increases. For example, for a model of 6 processes the proposed algorithm takes 1.386 seconds while using the finite automata algorithm over the reachability graph takes more than 326 seconds. Unfortunately it is also true that the execution time of the proposed algorithm grows exponentially as the number of processes increase for this particular Petri net model. Finally, it is worth noting that the proposed algorithm verifies a sufficient condition for the stability, while the finite automata approach verifies a sufficient and necessary condition.

In this chapter, a software tool that implements both the unfolding-based algorithm presented in chapter 4 and an algorithm that works over the reachability graph is presented. Also, a case study is described; it consists of a Petri net model for an implementation of the 2-phase commit protocol. The software tool was used to verify the model for up to 8 processes. The results show that for this particular case, the proposed unfolding-based algorithm is much faster than the analysis of the reachability graph.

Chapter 6 CONCLUSIONS

6.1. CONCLUDING REMARKS

On the topic of fault tolerant systems, the concept of state stability has been explored; it is a property related to the ability of a discrete event system to return to a desired state set in a finite number of steps.

While this property has been studied on finite automata, the issue has not received much attention on the Petri net community. However, in the case that the net structure can be used directly for analysis, greater efficiencies could be achieved.

In this thesis, research results regarding this issue were presented. In particular, a method to analyze the stability of a subclass of 1-bounded Petri nets has been proposed. The analysis algorithms are performed in polynomial time. Furthermore, an alternative method that determines the stability of 1-bounded Petri nets making use of the branching process representation of the net has been introduced. Finally, a bound to the number of steps required to return to the desired state set has been obtained; this bound is easy to calculate from a previously obtained subset.

Current and future research paths include finding the exact number of steps that are required for a system to go back into the desired state set and defining a method to synthesize controllers allowing recovering from unexpected events like faults.

6.2. FINDING THE MINIMUM VALUE OF K SO THAT THE SYSTEM IS K-STABLE

Once it is known that a system is returning to a desired state in a finite number of steps, it is useful to determine the maximum number of steps required to return to a desired state. In this sense, the property of k-stability is very useful: Given a complete 1-bounded IPN model Σ_f (free from undesired marking cycles) including the normal and unwanted behaviors and a IPN submodel Σ_n describing the normal behavior only, such that $\Sigma_n \subset \Sigma_f$, Σ_f is $k - stable$ if starting from any marking of Σ_f a marking also reachable on Σ_n is obtained within at most k steps.

A method to find a value of m such that the system is $k - stable$ for $k \geq m$ is presented in Chapter 4, section 5 of this document. It makes use of a subset MSS of the β^u subgraph ($|MSS| = m$). While the obtained value is useful as a bound for $k - stability$, it is possible that the system is $k - stable$ for some $k < m$; in other words, the obtained value is not the minimum value possible for which the system is $k - stable$.

One of the reasons why the obtained value is not the minimum is because the subset MSS used to calculate the value can be bigger than the set of events that is actually fireable on the original net. This is because MSS is not necessarily causally-closed or conflict-free (Def. 2.16), which are necessary restrictions for a set of events that is fireable from the initial marking.

When calculating the set , the size of the *MSS* that contains a cutoff event e also includes the size of the set of events that *would follow* e if it was not a cutoff event, this makes the size of *MSS* useful as a bound for $k - stability$. Of course, this size does not take into account that the set must be causally-closed and conflict-free. In other words, by adding the size of the set of events following a cutoff event to the size of *MSS* it is assumed that the events following the cutoff event are able to be fired without paying attention to additional preconditions needed by some of the events to fire. Unfortunately, it is not possible to obtain this information from the finite complete prefix; the nodes that follow the cutoff event are omitted from the FCP thus it is not possible to determine which nodes must be added to *MSS* for it to be causally-closed or if it is conflict free). One solution for this problem is to extend the finite complete prefix to include the events following a cutoff event as long as the new included events have the same marking as an event in β^u. If the system is *stable*, this extension operation should be finite. However, this approach is not necessarily more efficient than determining the property from the reachability graph. For this reason, the unfolding approach may not be the best way to solve the problem if the minimum value of k is required.

6.3. Finding Stabilizing Controllers

In this work we focused on methods for deciding if a system is stable. Other very important problem to address is finding a controller that makes a system stable when working in closed loop. Several approaches have been presented for finite automata modeled systems, the most important were presented in chapter 1.

Let $\Sigma_f = (P, T, I, O, \sigma, \lambda, \varphi, M_0) = (Q_f, M_0)$ be an 1-bounded IPN model where undesired events can occur, and $\Sigma_n = (P, T_n, I_n, O_n, \sigma, \lambda_n, \varphi_n, M_0) = (Q_n, M_0)$ be submodel of Σ_f that is free from undesired occurences, where $T_n \subset T$ $T - T_n = T_f$ such that T_f represents the undesired transitions, and $I_n(O_n)$, λ_n, φ_n are the restrictions of the corresponding functions to transitions in T_n.

Recall from 1.3.1 that a state feedback law is a mapping $K: R(Q, M_0) \rightarrow U$ where a reachable marking is mapped to an admissible control input (set of controllable events to disable in that marking), this mapping determines which controllable events remain enabled at a certain state.

Then, the objective is to find a control law $K: R(Q_f, M_0) \rightarrow U$ so that (Q_f, M_0) is stable with respect to $R(Q_n, M_0)$ while being working in closed-loop with K.

As a first approach to this problem, a method to enable an already controlled system to recover from a simple type of faults was presented in [Lutz-Ley, 2013a]. In this proposal, the control technique is based on an output tracking regulation approach similar to that of [Santoyo, 2008]. Basically, the system model is composed of a plant and a specification that are related between

them with bidirectional arcs as shown in figure 5.2. The strategy used in this method is to augment the specification to take into account fault occurrences.

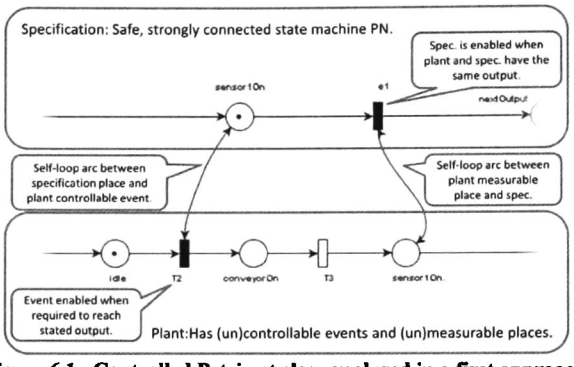


Figure 6.1. Controlled Petri net class analyzed in a first approach.

6.4.PUBLICATIONS

[Lutz-Ley, 2012a] A. Lutz-Ley, L.E. López-Mellado, "Stability-based Characterization of Fault Tolerance in Petri Net Models of Discrete Event Systems", Proc. of IEEE Conf. on Computational Engineering in Systems Applications (CESA'2012) pp.19-24. Santiago, Chile. April 2012.

[Lutz-Ley, 2012b] A. Lutz-Ley, L.E. López-Mellado, "Recoverability Analysis of Controlled Discrete Event Systems Modelled by a Class of Petri Nets", Int. Workshop on Discrete event systems (WODES'2012). Guadalajara, Mexico. Oct. 2012.

[Lutz-Ley, 2013a] A. Lutz-Ley, L.E. López-Mellado, "Synthesis of Fault Recovery Sequences in a Class of Controlled Discrete Event Systems Modelled With Petri Nets", 3rd Iberoamerican Conference on Electronics Engineering and Computer Science (CIIECC 2013) S.L.P, Mexico. April 2013.

[Lutz-Ley, 2013b] A. Lutz-Ley, L.E. López-Mellado, "State-Stability Analysis of Discrete Event Systems using Petri-net Branching Processes", 4th IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2013). York, United Kingdom, Sep. 2013.

[Lutz-Ley, 2015] A. Lutz-Ley, L.E. López-Mellado, "Stability Analysis of Discrete Event Systems Modeled by Petri Nets Using Unfoldings", Submitted to: IEEE Transactions on Automation Science and Engineering. 2015.

References

[Alcaraz 2006] M. Alcaraz-Mejia, E. Lopez-Mellado, A. Ramirez-Treviño , "Fault recovery of manufacturing systems based on controller reconfiguration," *System of Systems Engineering, 2006 IEEE/SMC International Conference on* vol., no., pp.6 pp., 24-26 April 2006

[Alcaraz 2007] M. Alcaraz-Mejia, E. Lopez-Mellado, A. Ramirez-Treviño, "A redundancy based method for Petri net model reconfiguration," *Systems, Man and Cybernetics, 2007.* ISIC. IEEE International Conference on , vol., no., pp.1382-1387, 7-10 Oct. 2007

[Arora, 1993] A. Arora, M. Gouda, "Closure and Convergence: A Foundation of Fault-Tolerant Computing", *IEEE Transactions on software engineering* Volume 19, No. 11, 1993.

[Astuti, 1995] Astuti P., McCarragher Brenan J., "The Convergence of Assembly Discrete Event Systems Using Markov Chains", *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on,* pp 1751-1756, May 1995.

[Astuti, 1996a] Astuti P., McCarragher Brenan J., "Discrete event controller synthesis for the convergence of an assembly process," *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on* , vol.2, no., pp.1153,1158 vol.2, 22-28 Apr 1996.

[Astuti, 1996b] Astuti P., McCarragher Brenan J., "The stability of a class of discrete event systems using Markov Chains", *International Journal of Control* , vol.64, Issue 3, 1996.

[Brave, 1990] Y. Brave, M. Heymann, "Stabilization of discrete-event processes", *International Journal of Control* Volume 51, Issue 5, 1990.

[Diekert, 1990] Diekert, V., "Combinatorics on Traces". LNCS, Vol. 454, Springer 1990.

[Dijkstra, 1974] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control", *Commun. ACM* 17, 11 , 643-644.

[Dingle, 2009] N.J. Dingle, W.J. Knottenbelt, T. Suto., "PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets", *ACM SIGMETRICS Performance Evaluation Review (Special Issue on Tools for Computer Performance Modelling and Reliability Analysis)*, Vol. 36(4), pp. 34-39, March 2009.

[Esparza, 1994] Esparza J., Nielsen M., "Decidability issues for Petri nets" Basic research in computer science BRICS 1994.

[Esparza, 2002] J. Esparza, S. Römer, W. "An improvement of McMillan's unfolding algorithm", *Formal methods in system design,* 20.

[Hillah, 2009] L.M. Hillah, E. Kindler, F. Kordon, L. Petrucci and N. Trèves, "A primer on the Petri Net Markup Language and ISO/IEC 15909-2" Petri Net Newsletter 76:9--28, October 2009. (originally presented at the 10th International workshop on Practical Use of Colored Petri Nets and the CPN Tools -- CPN'09)

[Khatab, 2002] A. Khatab, E.Niel, "State feedback stabilizing controller for the failure recovery of Timed Discrete Event Systems", *International Workshop on Discrete Event Systems*, 2002.

[Koh, 2009] I. Koh, F. DiCesare: "Modular transformation methods for generalized Petri nets and their application to automated manufacturing systems" *IEEE Transactions on Systems, Man, and Cybernetics* 21(6): 1512-1522 .1991

[Lutz-Ley, 2012a] A. Lutz-Ley, L.E. López-Mellado, "Stability-based Characterization of Fault Tolerance in Petri Net Models of Discrete Event Systems", Proc. of IEEE Conf. on Computational Engineering in Systems Applications (CESA'2012) pp.19-24. Santiago, Chile. April 2012.

[Lutz-Ley, 2012b] A. Lutz-Ley, L.E. López-Mellado, "Recoverability Analysis of Controlled Discrete Event Systems Modelled by a Class of Petri Nets", Int. Workshop on Discrete event systems (WODES'2012). Guadalajara, Mexico. Oct. 2012.

[Lutz-Ley, 2013a] A. Lutz-Ley, L.E. López-Mellado, "Synthesis of Fault Recovery Sequences in a Class of Controlled Discrete Event Systems Modelled With Petri Nets", 3rd Iberoamerican Conference on Electronics Engineering and Computer Science (CIIECC 2013) S.L.P, Mexico. April 2013.

[Lutz-Ley, 2013b] A. Lutz-Ley, L.E. López-Mellado, "State-Stability Analysis of Discrete Event Systems using Petri-net Branching Processes", 4th IFAC Workshop on Dependable Control of Discrete Systems (DCDS 2013). York, United Kingdom, Sep. 2013.

[McMillan, 1993] K.L McMillan, "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits", *Lecture Notes in Computer Science* Volume 663, 1993, pp 164-177

[Özveren, 1991] C.M Özveren, A.S. Willsky, P.J. Antsaklis, "Stability and Stabilizability of Discrete Event Dynamic Systems", *Journal of the Association for Computing Machinery*, Vol.38, No.3, 1991.

[Paoli, 2008] A. Paoli, M. Sartini, S. Lafortune, "A fault tolerant architecture for supervisory control of discrete event systems", *Proceedings of the 17th World Congress The International Federation of Automatic Control* Seoul, Korea, July 6-11, 2008

[Paoli, 2011] A. Paoli, M. Sartini, S. Lafortune, "Active fault tolerant control of discrete event systems using online diagnostics", *Automatica*, N.47 2011

[Passino, 1994] K.M. Passino, A.N. Michel, P.J. Antsaklis, "Lyapunov Stability of a Class of Discrete Event Systems", *IEEE Transactions on Automatic Control*, Vol.39, No.2, 1994.

[Passino, 1998] K.M. Passino, K.L. Burguess, "Stability Analysis of Discrete Event Systems", *Wiley-Interscience publication*, 1998.

[Retchkiman, 1999] Z. Retchkiman, "From Stability to the Stabilization problem of Discrete event systems modeled by Petri nets using Lyapunov methods", *American Control Conference*, 1999.

[Retchkiman, 2000] Z. Retchkiman, "A vector Lyapunov function approach for the stabilization of Discrete event systems", *American Control Conference*, 2000.

[Retchkiman, 2011] Z. Retchkiman, "The Stability Problem for Discrete Event Dynamical Systems Modeled with timed Petri Nets Using a Lyapunov-Max-Plus Algebra Approach", *International Mathematical Forum*, Vol. 6, 541-556, 2011.

[Rivera, 2004] I. Rivera, "Observability and modular synthesis of Petri net models of Discrete Event Systems", *Tesis Doctoral*, CINVESTAV-IPN Unidad Guadalajara, 2004.

[Rivera, 2005] I. Rivera, A. Ramirez, E. Lopez "Building Reduced Petri Net Models of Discrete Manufacturing Systems", *Mathematical and Computer Modeling*, Vol. 41, p923-937, 2005.

[Santoyo, 2008] A. Santoyo-Sanchez, A. Ramírez-Treviño, C. De Jesús-Velásquez, L.I. Aguirre-Salas, "Step State-feedback Supervisory Control of Discrete Event Systems using Interpreted Petri Nets", *IEEE International Conference on Emerging Technologies and Factory Automation*, (ETFA 2008). 2008

[Schneider, 1993] Y. Brave, M. Heymann, "Self-Stabilization", *ACM Computing Surveys* Volume 25, No.1, 1993.

[Tarraf, 2005] D.C. Tarraf, M.A. Dahleh, A.Megretski, "Stability of Deterministic Finite State Machines", *American Control Conference*, 2005.

[Tarraf, 2006] D.C. Tarraf, M.A. Dahleh, A.Megretski, "Input/Output Stability of Systems Over Finite Alphabets", *American Control Conference*, 2006.

[Wen, 2007] Q.Wen, R. Kumar, J.Huang, H.Liu, "Weakly Fault-Tolerant Supervisory Control of Discrete Event Systems", *American Control Conference*, 2007.

[Wen, 2008] Q. Wen, R. Kumar, J.Huang, H.Liu, "A Framework for Fault-Tolerant Control of Discrete Event Systems", *IEEE Transactions on Automatic Control*, Vol.53, No.8, 2008.

[Wen, 2009] Q. Wen, "Fault-tolerant supervisory control of discrete-event systems", *Tesis doctoral*, Iowa State University, 2009.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Estabilidad de Sistemas de Eventos Discretos Concurrentes modelados con redes de Petri. Stability of Concurrent Discrete Event Systems modeled as Petri nets.

del (la) C.

Alberto LUTZ LEY

el día 03 de Julio de 2015.

Dr. Félix Francisco Ramos Corchado
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

Dr. Antonio Ramírez Treviño
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González Pico
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dra. María Elena Meda Campaña
Profesor Titular
Universidad de Guadalajara CUCEA