REACHABILITY-BASED DIAGNOSABILITY FOR HYBRID SYSTEMS

Jéssica dos Santos Vieira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Elétrica, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Elétrica.

Orientadores: Eduardo Vieira Leão Nunes
Lilian Kawakami Carvalho

Rio de Janeiro
Junho de 2019

REACHABILITY-BASED DIAGNOSABILITY FOR HYBRID SYSTEMS

Jéssica dos Santos Vieira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA ELÉTRICA.

Examinada por:

RIO DE JANEIRO, RJ – BRASIL
JUNHO DE 2019

Vieira, Jéssica dos Santos

Reachability-based diagnosability for hybrid systems/Jéssica dos Santos Vieira. – Rio de Janeiro: UFRJ/COPPE, 2019.

XII, 75 p.: il.; 29, 7cm.

Orientadores: Eduardo Vieira Leão Nunes
Lilian Kawakami Carvalho

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia Elétrica, 2019.

Referências Bibliográficas: p. 64 – 66.

1. Diagnosis. 2. Hybrid systems. 3. Reachability. I. Nunes, Eduardo Vieira Leão *et al.* II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia Elétrica. III. Título.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

DIAGNOSE DE FALHAS DE SISTEMAS HÍBRIDOS BASEADA EM ALCANÇABILIDADE

Jéssica dos Santos Vieira

Junho/2019

Orientadores: Eduardo Vieira Leão Nunes
Lilian Kawakami Carvalho

Programa: Engenharia Elétrica

Neste trabalho apresentam-se os principais conceitos relacionados aos sistemas híbridos, que são sistemas dinâmicos que combinam comportamentos discreto e contínuo. Esses sistemas podem ser modelados por autômatos híbridos ou sistemas de transição. Autômatos híbridos permitem mais riqueza ao modelo, entretanto nem sempre é possível prever o seu comportamento. Dependendo do tipo de análise do sistema, como a verificação de certas propriedades, é preferível um maior nível de abstração sendo, assim, modelado por sistema de transição, os quais apresentam ferramentas mais estruturadas para verificação de propriedades. Este trabalho também apresenta uma nova definição de diagnosticabilidade que combina a diagnosticabilidade de sistemas a eventos discretos (SEDs) com a análise de alcançabilidade para comparação dos comportamentos contínuos. Além disso, apresenta-se um estudo de caso da análise da diagnosticabilidade de falhas de sistemas modelados por autômatos híbridos. Nesse exemplo demonstra-se a vantagem de se realizar a análise da alcançabilidade dos estados associados à dinâmica a tempo contínuo do modelo híbrido. Com essa abordagem, é possível diagnosticar falhas que não seriam possíveis usando técnicas puramente de SEDs.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

REACHABILITY-BASED DIAGNOSABILITY FOR HYBRID SYSTEMS

Jéssica dos Santos Vieira

June/2019

Advisors: Eduardo Vieira Leão Nunes
Lilian Kawakami Carvalho

Department: Electrical Engineering

In this work, we present the main concepts related to hybrid systems, that are dynamic systems that combine discrete and continuous behaviors. These systems can be modeled by hybrid automata or transition systems. Hybrid automata allow more wealth to the model, however, it is not always possible to predict its behavior. Depending on the type of analysis of the system, such as the verification of certain properties, a higher level of abstraction is preferred and, thus, modeled by a transition system, which presents more structured tools for checking properties. This work also presents a new definition of diagnosability combining the discrete event systems (DES) diagnosability with reachability analysis to compare continuous behaviors. Furthermore, we present a case study of fault diagnosability analysis of systems modeled by hybrid automata. This example demonstrates the advantage of performing the state reachability analysis associated with the continuous-time dynamics of the hybrid model. With this approach, it is possible to diagnose failures that would not be possible by using purely DES techniques.

Contents

List of Figures

Lista de Símbolos

A_l	Labeler automaton, p. 16
$Ac(G)$	Accessible part of automaton G, p. 10
$CoAc(G)$	Coaccessible part of an automaton G, p. 11
G	Deterministic automaton, p. 6
$G_1 \parallel G_2$	Parallel composition between automata G_1 and G_2, p. 14
$G_1 \times G_2$	Product composition between automata G_1 and G_2, p. 13
G_d	Diagnoser automaton, p. 17
G_{obs}	Observer automaton, p. 15
H	Hybrid automaton, p. 23
$L(G)$	Generated language by automaton G, p. 8
L/u	Post-language of L after a trace u, p. 5
$L_m(G)$	Marked language of automaton G, p. 8
$P(L)$	Projection operator of a language L, p. 6
$P^{-1}(L)$	Inverse projection operator of a language L, p. 6
TS	Transition system, p. 38
$Trim(G)$	Trim operation of automaton G, p. 11
$UR(x)$	Unobservable reach of a state x, p. 15
Σ^*	Kleene-closure of Σ, p. 5
Σ_o	Set of observable events, p. 5
Σ	Set of events, p. 4

Σ_{uo}	Set of unobservable events, p. 5
Φ	LTL formula, p. 43
$\Psi(\Sigma_f)$	Set of all sequences in L that end with event σ_f, p. 16
$\Sigma_s \notin u$	None of the events of Σ_s appears in u, p. 5
$\|u\|$	The length of a trace u, p. 4
$\mathcal{R}_H(Q_i, X_i)$	Reachable region for the set of discrete states Q_i and the set of continuous states X_i, p. 26
$\overline{L}$	Prefix-closure of L, p. 5
$\varepsilon R(x)$	$\varepsilon - reach$ of state x, p. 9
ε	Empty trace, p. 4

Chapter 1

Introduction

The technological evolution has brought great advances in industry which led to the concept of Industry 4.0. This new industrial revolution is based on cyber-physical systems operating in real time, decentralized and using the concept of *internet of things*. To meet this new industry requirements, the systems must run without interruption and if the system fails, this problem must be solved efficiently without causing major damage. Therefore, it is necessary that the fault diagnosis system be able to diagnose the failure in time so as not to cause damage neither to production nor to installation.

Fault diagnosis consists of determining whether the system is in its normal behavior or if any failure has occurred. One of the approaches to the fault diagnosis problem is based on the knowledge of the system model and, within this context, the works that use the so-called discrete event models [2, 3] stand out. However, the evolution of discrete event systems (DES) is due to the asynchronous occurrence of events and no information about the dynamic evolution of the system is used while the system remains in a certain state. Hybrid Systems (HS), on the other hand, are systems that have both continuous and discrete events based dynamics, which interact with each other during their evolution [4, 5]. Many DES represent an abstraction of HS, and depending on the level of that abstraction, information may be lost.

One of the most cited and a pioneer work is presented in HENZINGER [4], where it is proposed a formal definition for hybrid automata. A more continuous system approach is presented in GOEBEL *et al.* [6] and in FREHSE *et al.* [7] an abstraction refinement for hybrid systems is presented. Since HS theory is recent, there are not many consolidated textbooks that address the theme.

Fault diagnosis of HS is an incipient area with few published works. One approach found in the literature fault detection in finite-time using the invalidation model for affine switched systems is presented in HARIRCHI and OZAY [8] and in HARIRCHI *et al.* [9]. In this context, several models are obtained from the input/output observation of the system. These works are limited to continuous switched systems with no discrete event dynamics. Other works consider the fault diagnosis in HS as an extension of the fault diagnosis of DES by adding events through the discretization of the system analysis of the continuous dynamics [10, 11].

In BAYOUDH and TRAVÉ-MASSUYÈS [10], the idea is to abstract the continuous dynamics using a concept defined as signature-events associated with mode signatures and let the analysis be made in a DES structure. Each mode has its signature defined from the relation of its continuous dynamics with each of the other modes of the hybrid system. The signature events are created whenever the mode signature changes. The HS model is mapped into a DES infrastructure using mostly the DES tools for fault diagnosability verification.

More recently, DIENE *et al.* [1] present a new definition of diagnosability for hybrid systems, as well as a verification method based on the verifier automaton and the distinction of modes based on continuous states models through residual analysis. This work combines DES techniques with a classical approach to continuous systems, residual calculation, to distinguish modes and identify the faults.

Modeling a system is a relevant task. The correct modeling facilitates the analysis and verification of properties [12]. In this sense, model checking has many tools and can be used on a large scale. Among the various techniques used for HS, the *reachability analysis* stands out as one of the most relevant and with a direct application in the model checking of HS [13–15]. Efficient computational tools, such as SpaceEx [16, 17], are available for reachability analysis of HS subclasses.

In this work, we first present a bibliographic review of hybrid systems. In addition, we present a new definition of diagnosability combining the DES diagnosability by SAMPATH *et al.* [2] with reachability analysis to compare continuous behaviors. The reachability analysis is used to distinguish the different continuous behaviors of the modes of the underlying discrete model. In this manner, the failure can be detected in some systems. This may not be possible considering it as a purely discrete event model. Furthermore, we present a case study of fault diagnosability analysis of systems modeled by hybrid automaton inspired by a classic example of discrete event systems diagnosis to demonstrate the advantage of performing the state reachability analysis associated with the continuous-time dynamics of the hybrid model aiming at making the fault diagnosable. In addition we apply the definition proposed in this work to an existent HS diagnosability problem.

This work is structured as follows: In Chapter 2, we present some preliminary concepts and notations of DES. In Chapter 3, we present some fundamentals on HS, as well as some works in HS diagnosability field. In Chapter 4, we propose a new HS diagnosability definition, a case study, a comparison example and some final remarks. Most of the results presented in that Chapter 4 has been summarized in a paper [18]. In Chapter 5, we summarize our contributions to this work and present some possible future continuations.

Chapter 2

Discrete Event Systems

Discrete event systems are discrete event-based state systems, where the evolution of each state depends only on the occurrence of discrete asynchronous events over time. Event is an instant occurrence capable of causing a transition from one state to another. The concatenation of all possible event sequences of a system forms its language. These systems can be formally modeled by an automaton.

In this chapter, we present some preliminary concepts and notations related to DES theory. This chapter is based on the notations of CASSANDRAS and LAFORTUNE [19] and is structured into six sections: in Section 2.1 we introduce the notion of system language and some operations with languages; in Section 2.2, we present the formalism of the automaton, and the distinction among deterministic and nondeterministic automaton; in Section 2.3, we present the automata language; in Section 2.4 we present some operations performed on automata; in Section 2.5, we present the concept of observer automata, and, in Section 2.6, we present the concept of DES diagnosability.

2.1 Language

Let *alphabet* be a finite set of symbols Σ, usually associated with a set of physical events. A *trace* is a sequence of events taken out of this alphabet. For a trace u, $|u|$ denotes its length in number of events. The empty trace ε is a trace where there is no occurrence of events. Notice that the length of ε is zero.

Definition 1 (Language) *[19]*

A language L defined over a set of events Σ is a set of finite length traces formed from events in Σ.

Example 1 *Let $\Sigma = \{a, b, c\}$. Then, $L_1 = \{\varepsilon, a, cb, aaa, bcba\}$ and $L_2 = \{\varepsilon, c, bb, bbbca\}$ are languages defined over the set of events Σ. The length of trace cb is $|cb| = 2$ and the length of trace $bbbca$ is $|bbbca| = 5$.*

The *Kleene-closure* Σ^* denotes the set of all finite-length traces formed by the juxtaposition of symbols in a given alphabet Σ, including the empty trace, ε. Every language L is a subset of Σ^*, *i.e.*, $L \subseteq \Sigma^*$.

Example 2 *Let* $\Sigma = \{a, b\}$. *Thus, its Kleene-closure is given by the set* $\Sigma^* = \{\varepsilon, a, b, aa, ab, bb, aaa, aab, ...\}$.

The concatenation of two traces $u, v \in \Sigma^*$ is the combination of the trace u followed by the trace v, represented as uv. The concatenation of any traces $u, \varepsilon \in \Sigma^*$ is u itself, for ε is the *identity element.* A trace s can always be partitioned as $s = tuv$, where t is the prefix of s, and v is its suffix. It is important to notice that ε is always a prefix and suffix of any trace.

The concatenation of two languages L_1, $L_2 \subseteq \Sigma^*$ is $L_1L_2 := \{u \in \Sigma^* : (u = u_1u_2)$ and $(u_1 \in L_1)$ and $(u_2 \in L_2)\}$.

Example 3 *Let the languages* $L_1 = \{b, ac\}$ *and* $L_2 = \{\varepsilon, d, bb\}$. *The concatenation of* L_1 *and* L_2 *is given by* $L_1L_2 = \{b, bd, bbb, ac, acd, acbb\}$ *and the concatenation of* L_2 *and* L_1 *is given by* $L_2L_1 = \{b, ac, db, dac, bbb, bbac\}$.

The prefix closure of a language L, denoted by $\overline{L}$, consists of all prefixes of all traces in L. Formally $\overline{L} = \{u \in \Sigma^* : (\exists v \in \Sigma^*)[uv \in L]\}$. In general, $L \subseteq \overline{L}$ is said to be *prefix-closed* if $L = \overline{L}$.

Example 4 *Let the language* $L = \{a, bcba\}$. *The prefix-closure of* L *is given by* $\overline{L} = \{\varepsilon, a, b, bc, bcb, bcba\}$.

We define the post-language after a trace u in a language $L \subseteq \Sigma^*$, denoted by L/u as the set formed by the continuation of all traces of L after the occurrence of the trace u, *i.e.*, $L/u = \{v \in \Sigma^* : uv \in L\}$.

Example 5 *Let the language* $L = \{bcc, a, ac, bc, abc\}$. *The post-language after a trace* $u = bc$ *is* $L/u = \{c, \varepsilon\}$.

Let the trace $u \in \Sigma^*$ and the sub-alphabet $\Sigma_s \subseteq \Sigma$. The notation $\Sigma_s \notin u$ implies that there is no occurrence of the elements from Σ_s in u.

The set of events Σ can be partitioned in two disjoint subsets which are the observable event set Σ_o, and the unobservable event set Σ_{uo}, *i.e.*, $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Usually, observable events are associated to sensors.

Projection is an operation defined as $P : \Sigma^* \to \Sigma_o^*$, where $\Sigma_o \subseteq \Sigma$. It is recursively defined as $P(\varepsilon) = \varepsilon$; $P(\sigma) = \sigma$, if $\sigma \in \Sigma_o$, and $P(\sigma) = \varepsilon$, otherwise; and $P(u\sigma) = P(u)P(\sigma)$, for $u \in \Sigma^*$ and $\sigma \in \Sigma$. In a simplified way, the projection takes a trace formed by the set of events Σ and removes the events that do not belong to the set of events Σ_o. The inverse projection $P^{-1} : \Sigma_o^* \rightarrow 2^{\Sigma^*}$ is defined over $v \in \Sigma_o^*$ as $P^{-1}(v) = \{u \in \Sigma^* : P(u) = v\}$. Both projection and inverse projection operations can be extended to languages by applying them to all sequences in the language [19], $P(L) = \{t \in \Sigma_o^\star : (\exists s \in L)[P(s) = t]\}$ and $P^{-1}(L) = \{s \in \Sigma^\star : (\exists t \in L)[P(s) = t]\}$.

Example 6 *(Projection) Let us consider language $L_1 = \{a, c, bb, ac, bc, bcc, abc\}$ defined over $\Sigma = \{a, b, c\}$ and the set of observable events $\Sigma_o = \{a, b\} \subseteq \Sigma$. The projection $P : \Sigma^* \rightarrow \Sigma_o^*$ applied to language L_1 is the projection of each sequence in L, i.e., $P(L_1) = \{a, \varepsilon, bb, b, ab\}$.*

Example 7 *(Inverse Projection) Let us consider language $L_2 = \{\varepsilon, a, b, ab, bb\}$ defined over $\Sigma_o = \{a, b\} \subseteq \Sigma$, where Σ is defined as in example 6. The inverse projection $P^{-1} : \Sigma_o^* \rightarrow 2^{\Sigma^*}$ applied to language L_2 is presented as $P^{-1}(L_2) = \{c\}^* \cup \{c\}^*\{a\}\{c\}^* \cup \{c\}^*\{b\}\{c\}^* \cup \{c\}^*\{a\}\{c\}^*\{b\}\{c\}^* \cup \{c\}^*\{b\}\{c\}^*\{b\}\{c\}^*$.*

One way to graphically represent discrete event systems is by automata, which will be treated in Section 2.2.

2.2 Automata

An automaton is a device that is capable of representing a language according to well-defined rules [19]. It can be represented as a directed graph or a state transition diagram, where the vertices are the states and the edges represent the transitions from one state to another. Automata are classified as deterministic or nondeterministic.

2.2.1 Deterministic automata

A deterministic automaton is formally represented by a five-tuple $G = (X, \Sigma, f, x_0, X_m)$, where X is the state-space, Σ is the finite set of events, $f : X \times \Sigma \rightarrow X$ is the transition function, $x_0 \in X$ is the initial state of the system, and $X_m \subseteq X$ is the set of marked states, usually states which are important in the system. We can also define $\Gamma : X \rightarrow 2^\Sigma$ as the feasible event function in a state of G, where $\Gamma(x)$ is the set of all events σ for which $f(x, \sigma)$ is defined.

Example 8 *Let us consider a deterministic automaton G represented in Figure 2.1. For this automaton we can identify the following elements: the state-space $X = \{x, y, z\}$, the set of events $\Sigma = \{a, b, c\}$, the transition function f given by $f(x, a) = f(x, b) = z$, $f(x, c) = y$, $f(y, b) = y$ and $f(z, a) = y$, the initial state $x_0 = x$ and the set of marked states $X_m = \{y\}$. The states are represented by*

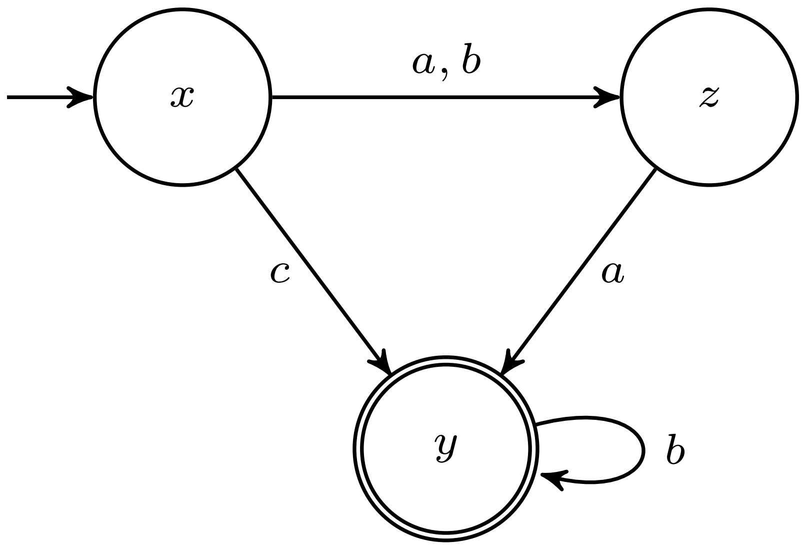


Figure 2.1: State transition diagram of automaton G of Example 8.

circles, the transitions are represented by an arrow that goes from the origin state $x_i \in X$ to state $f(x_i, \sigma)$ labeled with event $\sigma \in \Sigma$. The marked state is represented by a double circle, and the initial state is marked by an arrow pointing to it.

2.2.2 Nondeterministic automata

A nondeterministic automaton is formally represented by a five-tuple $G_{nd} = (X, \Sigma \cup \{\varepsilon\}, f_{nd}, X_0, X_m)$. It differs from the deterministic automaton in two aspects: nondeterministic transition function $f_{nd} : X \times \Sigma \cup \{\varepsilon\} \rightarrow 2^X$, *i.e.*, the transition function can evolve to more than one state and the initial state X_0 may be a subset of X, $X_0 \subseteq X$. Note that a transition may be labeled with empty trace ε in this automaton.

Example 9 *Let us consider a nondeterministic automaton G_{nd} represented in Figure 2.2. For this automaton we can identify the following elements: the state-space $X = \{v, w, x, y, z\}$, the set of events $\Sigma = \{a, b, c\}$, the nondeterministic transition function f_{nd}, which is given by $f_{nd}(w, a) = \{y\}$, $f_{nd}(w, c) = \{v\}$, $f_{nd}(v, \varepsilon) = \{z\}$, $f_{nd}(x, b) = \{z\}$, $f_{nd}(x, c) = \{y, z\}$, $f_{nd}(y, b) = \{y\}$ and $f_{nd}(z, a) = \{y\}$, the set of initial states $X_0 = \{x, w\}$, and the set of marked states $X_m = \{y\}$.*

2.3 Automaton language

The language of an automaton represents all sequences of events that the automaton is able to generate. It can be obtained by following all of the directed paths in the state transition diagram. Each automaton is related with two languages: the generated language and the marked language. For the following definitions, is

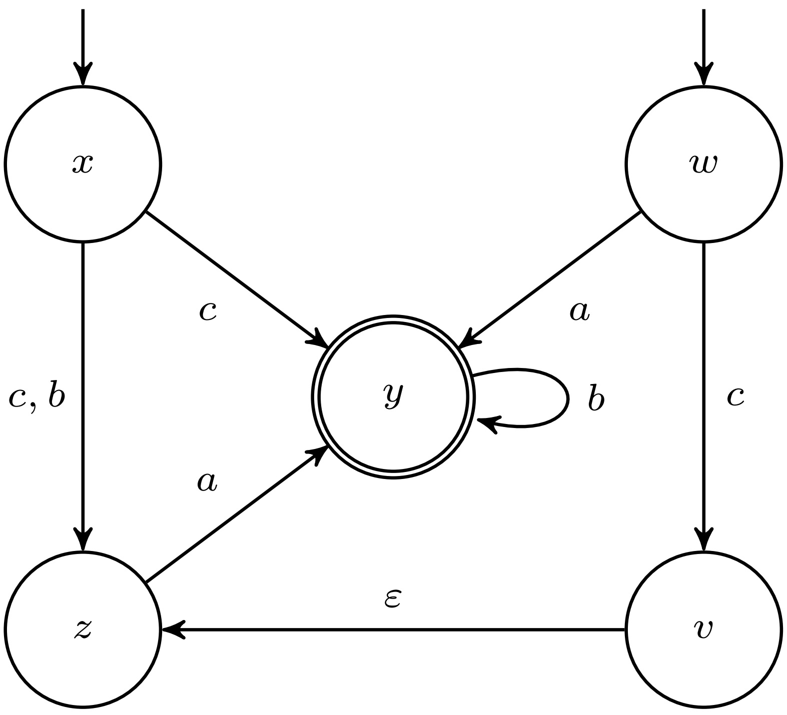


Figure 2.2: State transition diagram of a nondeterministic automaton G_{nd} of Example 9.

important to assume the extended transition function of deterministic automaton $f : X \times \Sigma^* \to X$ as follows.

$$\begin{aligned} f(x, \varepsilon) &:= x, \\ f(x, s\sigma) &:= f[f(x, s), \sigma] \text{ for } s \in \Sigma^* \text{ and } \sigma \in \Sigma. \end{aligned}$$

Definition 2 *The language generated by a deterministic automaton* $G = (X, \Sigma, f, x_0, X_m)$ *is:*

$$L(G) := \{s \in \Sigma^* : f(x_0, s) \text{ is defined }\},$$

and the marked language of a deterministic automaton G is:

$$L_m(G) := \{s \in \Sigma^* : f(x_0, s) \in X_m\}.$$

Example 10 *The generated and marked languages of automaton G, depicted in Figure 2.1, are* $L(G) = \{\varepsilon, a, b, c, aa, ba, cb, aab, bab, cbb, ...\}$ *and* $L_m(G) = \{c, aa, ba, cb, aab, bab, cbb, aabb, babb, cbbb, ...\}$*, respectively.*

Before defining the generated and marked language for the nondeterministic automaton, we need to extend the nondeterministic transition function, denoted by

f_{nd}^{ext}, to the domain $X \times \Sigma^*$. Differently from the deterministic automaton, where $f(x, \varepsilon) = x$, in a nondeterministic automaton, the ε event may lead to a different state. For that purpose, we start defining the *ε-reach* of a state x, denoted by $\varepsilon R(x)$, which is the set of all states, including x, that can be reached from x by following transitions labeled with ε. This function may consider a set of states $B \subseteq X$, and it is defined by $\varepsilon R(B) := \bigcup_{x \in B} \varepsilon R(x)$.

The transition function for nondeterministic automata can be constructed recursively as follows:

$$f_{nd}^{ext}(x, \varepsilon) := \varepsilon R(x),$$

and for $u \in \Sigma^*$, and $\sigma \in \Sigma$:

$$f_{nd}^{ext}(x, u\sigma) := \varepsilon R(z : z \in f_{nd}(y, \sigma) \text{ for some state } y \in f_{nd}^{ext}(x, u)).$$

Definition 3 *The language generated by a nondeterministic automaton $G_{nd} = (X, \Sigma \cup \{\varepsilon\}, f_{nd}, x_0, X_m)$ is:*

$$L(G_{nd}) := \{s \in \Sigma^* : f_{nd}^{ext}(x_0, s) \textit{ is defined } \},$$

and the marked language of a nondeterministic automaton G_{nd} is given by:

$$L_m(G_{nd}) := \{s \in \Sigma^* : f_{nd}^{ext}(x_0, s) \cap X_m \neq \emptyset\}.$$

Example 11 *The generated and marked languages of automaton G_{nd}, shown in Figure 2.2, are, respectively, $L(G_{nd}) = \{\varepsilon, a, b, c, ab, ba, ca, cb, abb, bab, cab, cbb, ...\}$ and $L_m(G_{nd}) = \{a, c, ab, ba, ca, cb, abb, bab, cab, cbb, abbb, babb, cabb, cbbb, ...\}$.*

Remark 1 *The generated language L for the deterministic and nondeterministic automaton is prefix-closed, i.e., $\overline{L}$.*

2.4 Operations on automata

Automata can perform several operations that help to properly model a system. Certain operations are performed in a single automaton, the unary operations, and others are combination of two or more automata, the composition operations. Both operations are covered in this subsection.

2.4.1 Accessible part

The accessible part of an automaton G with generated language $L(G)$ and marked language $L_m(G)$ results in an automaton formed by all states that can be reached

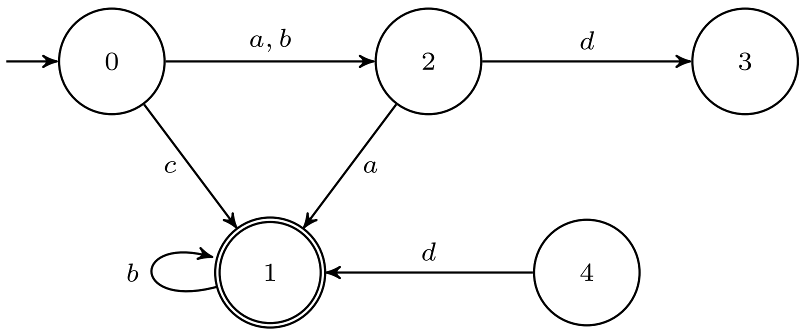


Figure 2.3: Automaton G of Example 12

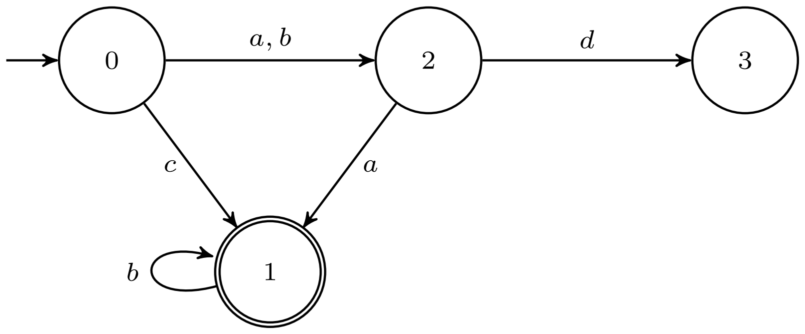


Figure 2.4: Accessible part of automaton G of Example 12.

from the initial state x_0. This operation is denoted as $Ac(G)$, formally:

$$Ac(G) \;:=\; (X_{ac}, \Sigma, f_{ac}, x_0, X_{ac,m}),$$

where

$$\begin{aligned} X_{ac} &= \{x \in X : (\exists s \in \Sigma^*)[f(x_0, s) = x]\} \\ X_{ac,m} &= X_m \cap X_{ac} \\ f_{ac} &= f\,|_{X_{ac}\times\Sigma\to X_{ac}} \end{aligned}$$

where $f\,|_{X_{ac}\times\Sigma\to X_{ac}}$ denotes the transition function f restricted to domain X_{ac}.

Example 12 *Let us consider automaton G, represented in Figure 2.3. The state transition diagram of the accessible part of automaton G is represented in Figure 2.4. There is no sequence from state 0 that leads to state 4; therefore, this state does not belong to $Ac(G)$.*

Notice that the accessible part operation does not change the generated $L(G)$ and marked $L_m(G)$ languages.

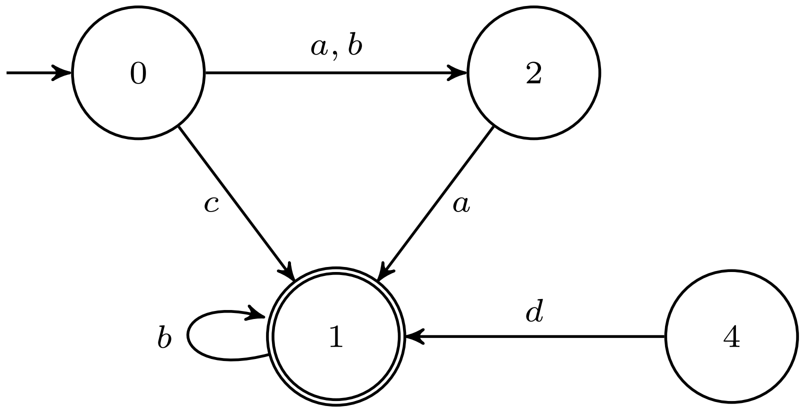


Figure 2.5: Coaccessible part of automaton G of Example 13.

2.4.2 Coaccessible part

The coaccessible part of an automaton G, with generated language $L(G)$ and marked language $L_m(G)$, results in an automaton formed by all states from which a marked state can be reached. This operation is denoted as $CoAc(G)$. Formally:

$$CoAc(G) \quad := \quad (X_{coac}, \Sigma, f_{coac}, x_{0coac}, X_m),$$

where

$$\begin{aligned} X_{coac} &= \{x \in X : (\exists s \in \Sigma^*)[f(x,s) \in X_m]\} \\ x_{0coac} &= \begin{cases} x_0, \text{ if } x_0 \in X_{coac} \\ \text{undefined, otherwise} \end{cases} \\ f_{coac} &= f\mid_{X_{coac} \times \Sigma \to X_{coac}} \end{aligned}$$

Example 13 *Let us consider the automaton G, represented in Figure 2.3. The state transition diagram of the coaccessible part of automaton G is represented in Figure 2.5. There is no sequence from* 3 *that leads to the marked state* 1*; therefore, this state does not belong to* $CoAc(G)$.

2.4.3 Trim operation

The trim operation of an automaton G is obtained by taking both the accessible part and the coaccessible part of G. This operation is denoted by $Trim(G)$, *i.e.*, $Trim(G) = Ac(CoAc(G)) = CoAc(Ac(G))$. An automaton G is called a trim automaton if $G = Trim(G)$.

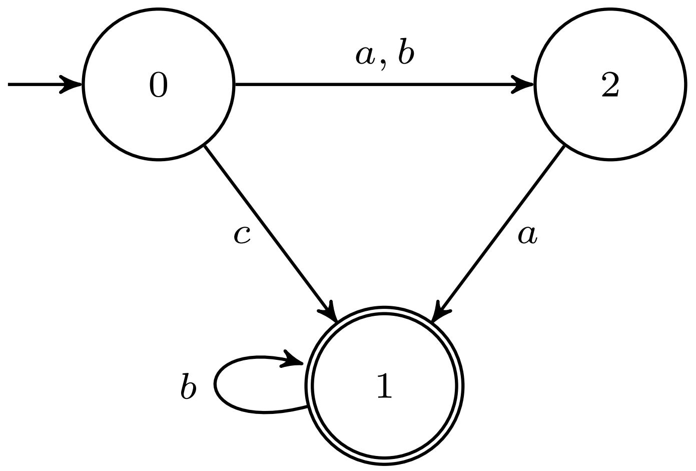


Figure 2.6: Trim operation of automaton G of Example 14.

Example 14 *Let us consider the automaton G, represented in Figure 2.3. The state transition diagram obtained by the trim operation over automaton G is represented in Figure 2.6.*

Remark 2 *The operations of taking the accessible and the coaccessible part of an automaton $G = (X, \Sigma, f, x_0, X_m)$ do not change the set of events Σ of the resulting automaton.*

Remark 3 *The accessible and coaccessible part and the trim operation can be performed similarly for nondeterministic automata.*

There are two different composition operations on automata, the product and the parallel composition. These operations model automata that operate concurrently.

2.4.4 Product composition

The product composition or completely synchronous composition is denoted by $\times$. In the product composition, an event occurs if, and only if, it is active in both automaton states simultaneously. For two automata G_1 and G_2, $G_1 \times G_2$ denotes the product composition between them. The intersection of two languages can be obtained by performing the product of their automaton representations. The product composition of automata $G_1 = (X_1, \Sigma_1, f_1, x_{01}, X_{m1})$ and $G_2 = (X_2, \Sigma_2, f_2, x_{02}, X_{m2})$ is given by:

$$\begin{aligned} G_1 \times G_2 &:= Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\times 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}), \text{where} \\ f_{1\times 2}((x_1, x_2), \sigma) &:= \begin{cases} (f_1(x_1, \sigma), f_2(x_2, \sigma)), \text{ if } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ \text{undefined, otherwise.} \end{cases} \end{aligned}$$

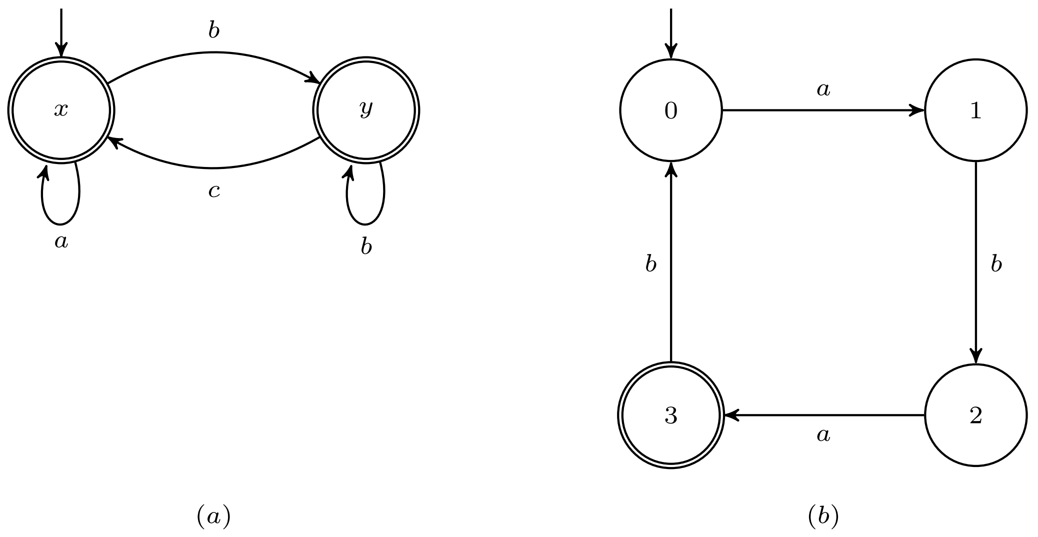


Figure 2.7: Automata G_1 and G_2 for Examples 15 and 16.

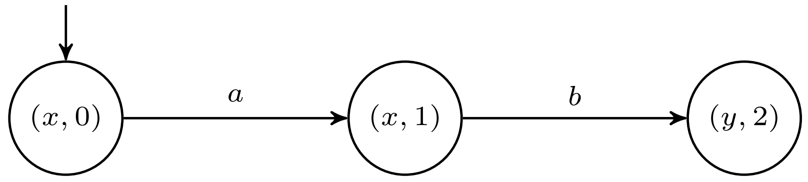


Figure 2.8: Product composition of G_1 and G_2 of Example 15.

where Γ_1 and Γ_2 are the feasible event functions in a state of G_1 and G_2, respectively.

Example 15 *Let us consider automata G_1 and G_2 represented in Figure 2.7 (a) and (b), respectively, where $\Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{a, b\}$. The product composition of G_1 and G_2 is shown in Figure 2.8.*

2.4.5 Parallel composition

The parallel composition or synchronous composition is denoted by $\|$. In the parallel composition, a common event between two automata can only occur if it is active in both automata simultaneously. An event that is not shared by the other automaton is called *private* and it can occur whenever it is active in a state. The parallel composition of automata $G_1 = (X_1, \Sigma_1, f_1, x_{01}, X_{m1})$ and $G_2 = (X_2, \Sigma_2, f_2, x_{02}, X_{m2})$ is given by:

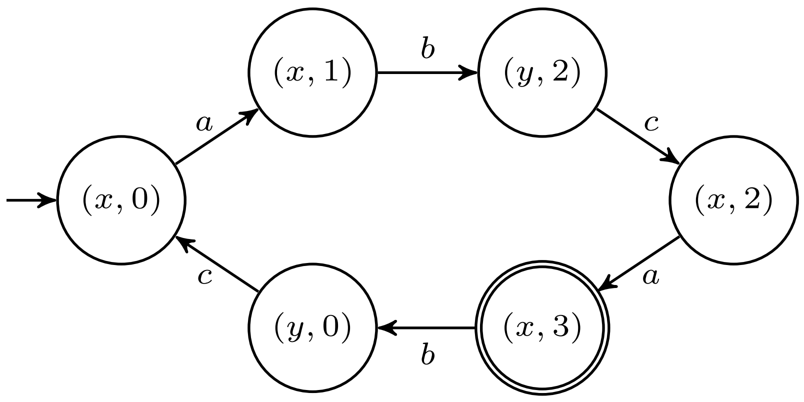

$$\begin{tikzcd}
 & (x,1) \arrow[r, "b"] & (y,2) \arrow[dr, "c"] & \\
\arrow[r] & (x,0) \arrow[u, "a"] & & (x,2) \arrow[dl, "a"] \\
 & (y,0) \arrow[ul, "c"] & \boxed{(x,3)} \arrow[l, "b"] &
\end{tikzcd}$$

Figure 2.9: Parallel composition of G_1 and G_2 of Example 16.

$$
\begin{aligned}
G_1 \parallel G_2 &:= Ac(X_1 \times X_2, \Sigma_1 \cup \Sigma_2, f_{1\parallel 2}, (x_{01}, x_{02}), X_{m1} \times X_{m2}), \text{ where} \\
f_{1\parallel 2}((x_1, x_2), \sigma) &= \begin{cases} (f_1(x_1,\sigma), f_2(x_2,\sigma)) \text{ if, } \sigma \in \Gamma_1(x_1) \cap \Gamma_2(x_2) \\ (f_1(x_1,\sigma), x_2) \text{ if, } \sigma \in \Gamma_1(x_1) \backslash \Sigma_2 \\ (x_1, f_2(x_2,\sigma)) \text{ if, } \sigma \in \Gamma_2(x_2) \backslash \Sigma_1 \\ \text{undefined, otherwise.} \end{cases}
\end{aligned}
$$

Example 16 *Let us consider automata G_1 and G_2 represented in Figure 2.7 (a) and (b), respectively, where $\Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{a, b\}$. The parallel composition of G_1 and G_2 is shown in Figure 2.9.*

Remark 4 *The product operation involves only events in $\Sigma_1 \cap \Sigma_2$ and the parallel operation involves events in $\Sigma_1 \cup \Sigma_2$. The resultant automaton of both operations has $\Sigma_1 \cup \Sigma_2$ as the set of events.*

Remark 5 *The product and parallel operations are associative. Consider automata G_1, G_2 and G_3, we define:*

$$
\begin{aligned}
G_1 \times G_2 \times G_3 &:= (G_1 \times G_2) \times G_3 = G_1 \times (G_2 \times G_3) \\
G_1 \parallel G_2 \parallel G_3 &:= (G_1 \parallel G_2) \parallel G_3 = G_1 \parallel (G_2 \parallel G_3)
\end{aligned}
$$

2.5 Observer automata

There may exist events whose occurrence are not detected by the system because there are no sensors to register them. Such events are called unobservable events. The events whose occurrence can be detected by a sensor, are called observable. Let us consider that the set of events can be partitioned as $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$, where Σ_o and

Σ_{uo} are the sets of observable and unobservable events of the system, respectively. The ε-transition is considered an unobservable event [19, 20]. In order to construct the observer automaton, denoted as G_{obs}, it is important the notion of unobservable reach of a state $x \in X$, which is a generalization of the notion of $\varepsilon - reach$.

Definition 4 (Unobservable reach) *The unobservable reach of a state $x \in X$, denoted by $UR(x)$, is defined as:*

$$UR(x) = \{y \in X : (\exists t \in \Sigma_{uo}^*)[f(x,t) = y]\}. \tag{2.1}$$

The unobservable reach can also be extended to sets of states $B \in 2^X$ as:

$$UR(B) = \bigcup_{x \in B} UR(x). \tag{2.2}$$

The computation of $G_{obs} = (X_{obs}, \Sigma_o, f_{obs}, x_{0,obs}, X_{m,obs})$ can be obtained by the following algorithm [19, 20].

Algorithm 1 *Observer automaton construction*

```
Input: $G = (X, \Sigma, f, x_0, X_m)$.
Output: Observer automaton $G_{obs} = (X_{obs}, \Sigma_o, f_{obs}, x_{0,obs}, X_{m,obs})$.

    Step 1: Define $x_{0,obs} = UR(x_0)$. Set $X_{obs} = \{x_{0,obs}\}$ and $\widetilde{X}_{obs} = X_{obs}$.
    Step 2: $\widehat{X}_{obs} = \widetilde{X}_{obs}$, $\widetilde{X}_{obs} = \emptyset$.
    Step 3: For each $B \in \widehat{X}_{obs}$ do
        3.1: $\Gamma_{obs}(B) = \left(\bigcup_{x \in B} \Gamma(x)\right) \cap \Sigma_o$.
        3.2: For each $\sigma \in \Gamma_{obs}(B)$,
                 $f_{obs}(B,\sigma) = UR(\{x \in X : (\exists y \in B)[x = f(y,\sigma)]\})$.
        3.3: $\widetilde{X}_{obs} \leftarrow \widetilde{X}_{obs} \cup f_{obs}(B,\sigma)$.
    Step 4: $X_{obs} \leftarrow X_{obs} \cup \widetilde{X}_{obs}$.
    Step 5: Repeat steps 2 to 4 until all accessible part of $G_{obs}$ is constructed.
    Step 6: $X_{m,obs} = \{B \in X_{obs} : B \cap X_m \neq \emptyset\}$
```

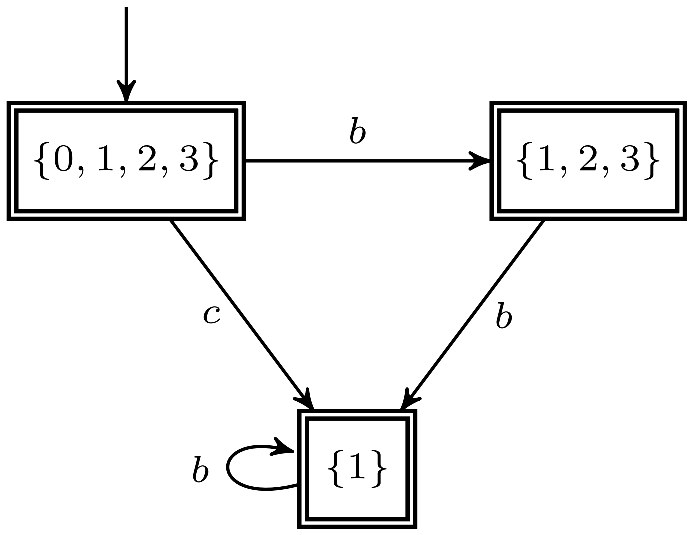


Figure 2.10: Automaton G_{obs} of Example 17

Example 17 *Let us consider automaton G depicted in Figure 2.4, with the set of observable events $\Sigma_o = \{b, c\}$ and initial state $x_0 = 0$. The observer automaton G_{obs} of automaton G is represented in Figure 2.10 and is constructed as in algorithm 1. The feasible event function $\Gamma(0) = \{a, b, c\}$ contains the unobservable event a, thus the system can transition to states 1 and 2 without being detected. Once in state 2, it can transition to states 1 and 3 without being noticed, where $\Gamma(2) = \{a, d\} \in \Sigma_{uo}$; therefore $x_{0,obs} = UR(x_0) = \{0, 1, 2, 3\}$. In state $x_{0,obs}$ can occur events $\Gamma_{obs}(x_{0,obs}) = \{b, c\}$, where $f(0, b) = \{2\}$ and $UR(2) = \{1, 3\}$, $f(1, b) = \{1\}$, therefore $f_{obs}(x_{0,obs}, b) = \{1, 2, 3\}$; $f(0, c) = 1$, therefore $f_{obs}(x_{0,obs}, c) = \{1\}$; $\Gamma(1) = \{b\}$ and $f(1, b) = 1$; $\Gamma_{obs}(\{1, 2, 3\}) = \{b\}$, $f(1, b) = \{1\}$, $f_{obs}(\{1, 2, 3\}, b) = \{1\}$. Notice all sates in G_{obs} are marked, since the marked state 1 in G is present in all states of G_{obs}.*

2.6 DES diagnosability

A fault event σ_{f_i}, $i = 1, 2, \ldots, n$, is the event responsible for leading the system to a fault state. Let Σ_f be the set of fault events. Therefore, $\Sigma_f = \{\sigma_{f_1}, \sigma_{f_2}, \ldots, \sigma_{f_n}\}$. When Σ_f is a single element set, we note σ_{f_i} as σ_f. We will consider all fault events as unobservable events, *i.e.*, $\Sigma_f \subseteq \Sigma_{uo}$, since an observable fault implies in trivial identification. Consider s_f as the last event of a sequence s, $\Psi(\Sigma_f)$ as the set of all sequences in L that end with event σ_f, *i.e.*, $\Psi(\Sigma_f) = \{s \in L : s_f \in \Sigma_f\}$ and $L/s = \{t \in \Sigma^* : st \in L\}$ as the continuation of language L after a sequence s. With abuse of notation, we say that a sequence $s \in L$ contains a fault event if $\Sigma_f \in s$[20].

The formal definition for DES diagnosability is provided by SAMPATH *et al.*

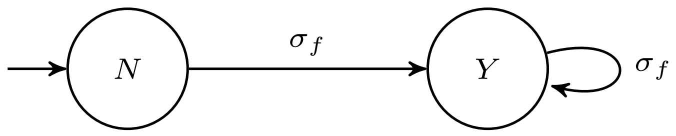


Figure 2.11: Labeler automaton A_l

[2], as follows.

Definition 5 *Let L be a language generated by an automaton G and assume that L is live and prefix-closed. Then L is diagnosable with respect to the projection P and $\Sigma_f = \{\sigma_f\}$ if the following condition is verified [20]:*

$$(\exists n \in \mathbb{N})(\forall s \in \Psi(\Sigma_f))(\forall t \in L/s)(|t| \geq n \implies D)$$

where D is the diagnosability condition

$$D = (\forall \omega \in P^{-1}(P(st)) \cap L)(\Sigma_f \in \omega)$$

In words, the language L is diagnosable if there exists an arbitrarily long sequence after the occurrence of a fault event whose projection is different from all normal sequence projection.

One way of verifying diagnosability of the generated language of an automaton G is with the diagnoser automaton G_d, which is a tool to perform the fault diagnosis of a system, where the diagnoser automaton is represented by

$$G_d = \{X_d, \Sigma_o, f_d, x_{0,d}\}$$

The diagnoser automaton G_d is obtained by performing the parallel composition with the *labeler automaton* A_l, represented in Figure 2.11 and then calculating the observable automaton from the previous operation,*i.e.*, $G_d = Obs(G \parallel A_l)$.

We call uncertain states, states of G_d that contains at least one state of G labeled with Y and at least one state of G labeled with N.

Definition 6 *A set of uncertain sates $\{x_{d_1}, x_{d_2}, \cdots, x_{d_p}\} \subset X_d$ forms an indeterminate cycle if the following conditions are satisfied[20]:*

(i) $x_{d_1}, x_{d_2}, \cdots, x_{d_p}$ *forms a cycle in G_d;*

(ii) $\exists ({x_l}^{k_l}, Y), (\tilde{x}_l^{r_l}, N) \in x_{d_l}, {x_l}^{k_l}$ *not necessarily distinct from* $\tilde{x}_l^{r_l}$, $l = 1, 2, \cdots, p$, $k_l = 1, 2, \cdots, m_l$ *and* $r_l = 1, 2, \cdots, \tilde{m}_l$ *such that the sequences of states* $\{{x_l}^{k_l}\}$ *and* $\{\tilde{x}_l^{r_l}\}$ *can be rearranged to form cycles in G, whose correspondent sequences s and $\tilde{s}$, formed with events that define the cycles evolution, have as projection* $\sigma_1\sigma_2\cdots\sigma_p$ *defined as the item above.*

Theorem 1 *The generated language L of an automaton G is diagnosable with respect to projection P and $\Sigma_f = \sigma_f$, if and only if, its diagnoser G_d does not contain indeterminate cycles [20].*

Example 18 *Let us consider automaton G depicted in Figure 2.12(a), with the set of observable events $\Sigma_o = \{a, b\}$. The only unobservable event is also the fault event σ_f, in this example, $\Sigma_f = \{\sigma_f\}$ and $\Sigma_f = \Sigma_{uo}$. Figures 2.12(b) and (c) represent the parallel composition $G \parallel A_l$ and the diagnoser automaton $G_d = Obs(G \parallel A_l)$, respectively. We can simplify the notation for the states of the diagnoser, as depicted in Figure 2.12(c). Instead of $(1, N)$ and $(0, Y)$, we may note as $\{1N, 0Y\}$.*

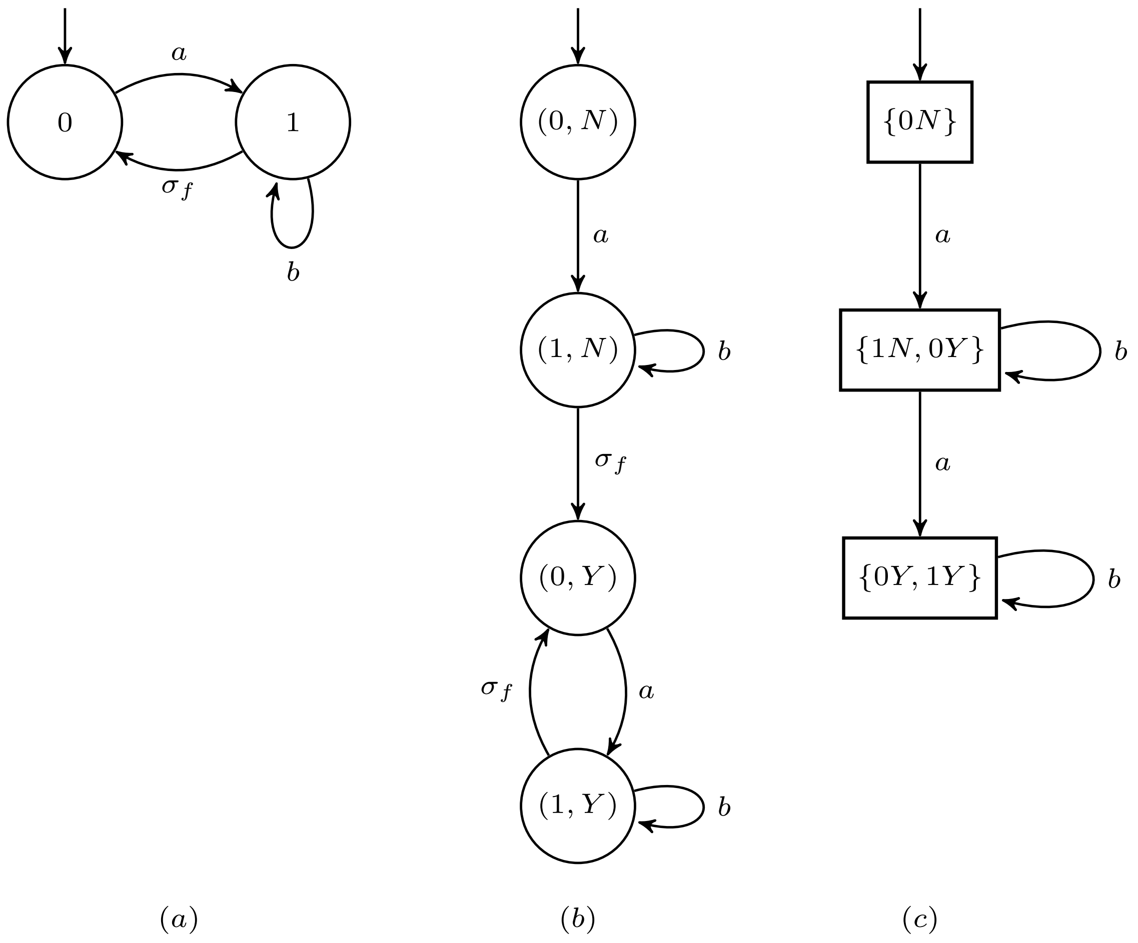


Figure 2.12: Automata G (a); parallel composition $G \parallel A_l$(b); diagnoser automaton $G_d = Obs(G \parallel A_l)$ of Example 18.

Notice, in Example 18, that there is an uncertain cycle $\{1N, 0Y\}$, however it is possible to confirm that in $G \parallel A_l$ this cycle exists only in $1N$. For this reason, it is not an indeterminate cycle and, therefore, the language generated by G is diagnosable.

Example 19 *Let us consider automaton G depicted in Figure 2.13(a), with the set of observable events $\Sigma_o = \{a, c\}$ and the fault event σ_f, $\Sigma_f = \{\sigma_f\}$ and $\Sigma_f \subseteq \Sigma_{uo}$. Figures 2.13(b) and (c) depict the parallel composition $G \parallel A_l$ and the diagnoser automaton $G_d = Obs(G \parallel A_l)$, respectively.*

Notice, in Example 19, that there is an uncertain cycle formed by $\{2N, 2Y\}$. Both cycles occur in $G \parallel A_l$, *i.e.* both normal $s_N = abcc^*$ and fault $s_F = \sigma_f acc^*$ sequences have the same projection, $P(s_N) = P(s_F) = acc^*$. For that reason, the language generated by G is not diagnosable.

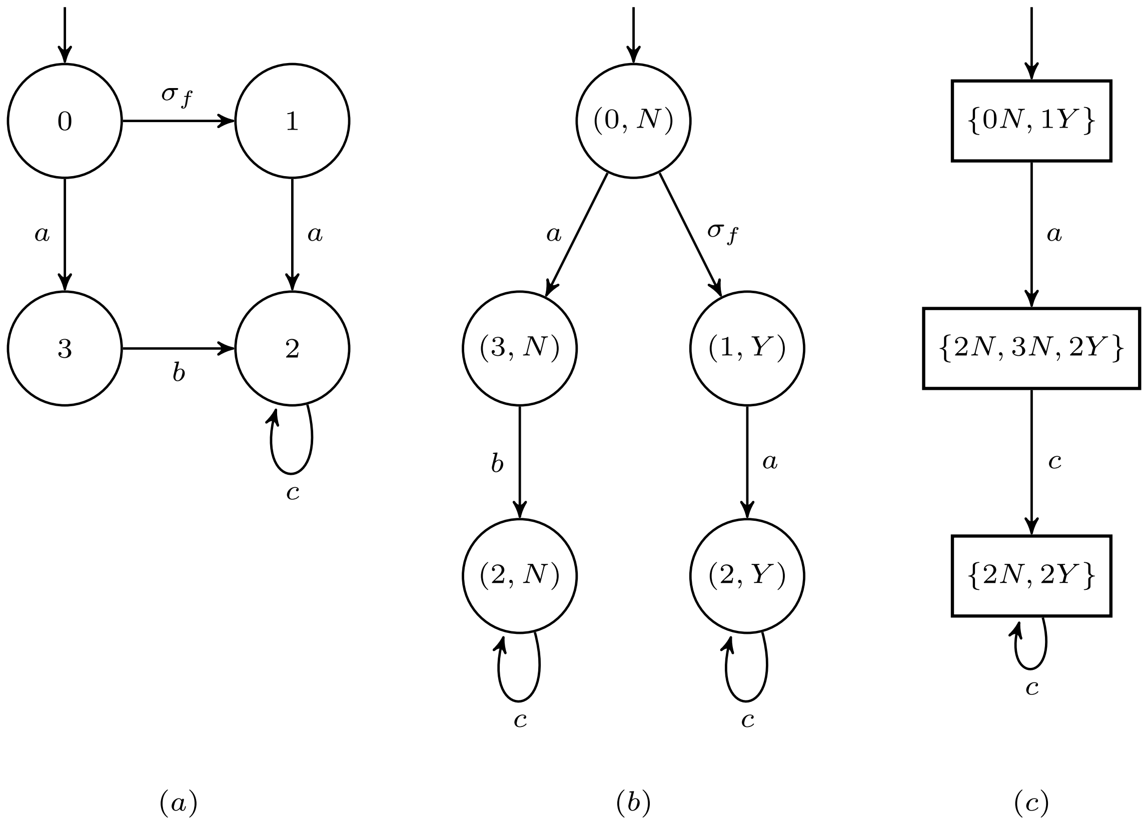


Figure 2.13: Automaton G (a); parallel composition $G \parallel A_l$(b); diagnoser automaton $G_d = Obs(G \parallel A_l)$ of Example 19.

2.6.1 New necessary and sufficient condition for DES diagnosability

The method proposed by SAMPATH *et al.* [2] has high computational cost with respect to the search for cycles. A recent work [21] proposes a new method based on the search for strongly connected components (SCCs), which is less costly. A set of vertices V_{SCC} of a direct graph $D = (V, E)$ is a SCC of D if it is a maximal set and all pairs of vertices u, v in V_{SCC} are reached form each other, where V is the

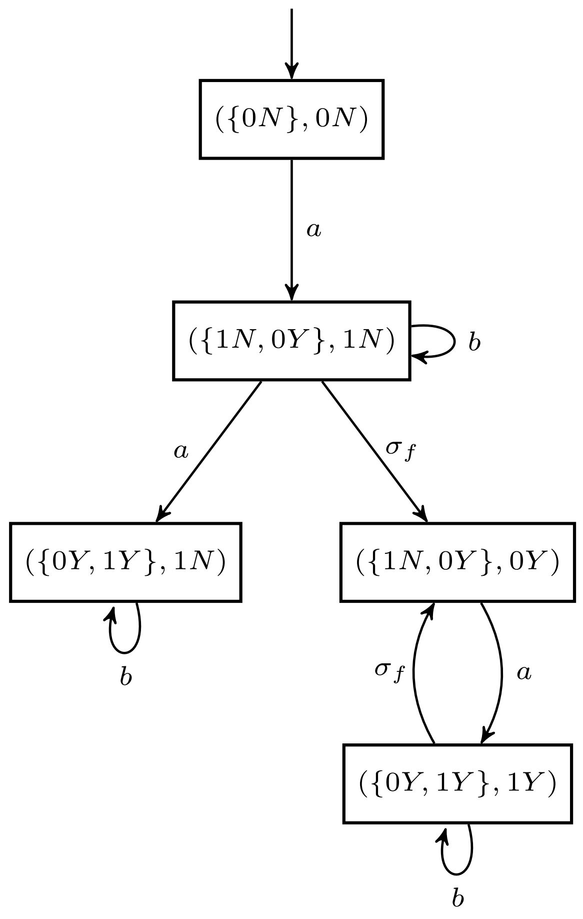


Figure 2.14: Automaton $G_{SCC} = G_d||G_\ell$ of Example 20

set of vertices and E is the set of edges of D. A nontrivial SCC is a singleton set, *i.e.*, V_{SCC} contains a single vertice, with a self-loop.

Consider automaton $G_\ell = G \parallel A_\ell$ and the new diagnoser automaton $G_{SCC} = G_d \parallel G_\ell$, a necessary and sufficient condition for language diagnosability is presented in Lemma 1.

Lemma 1 *The language L generated by automaton G is diagnosable with respect to projection $P : \Sigma^* \implies \Sigma_o^*$ and $\Sigma_f = \{\sigma_f\}$ if, and only if, G_{SCC} does not have nontrivial SCCs formed with states (x_d, x_ℓ), such that x_d is uncertain and x_ℓ is a Y-labeled state.*

In words, a language is not diagnosable if, and only if, it contains a state with a self-loop, where the state is composed by an uncertain state of G_d and a Y-labeled state of G_ℓ.

Example 20 *Consider automata G_d and G_ℓ of Example 18 depicted in Figure 2.12 (c) and (b), respectively. The diagnoser automaton $G_{SCC} = G_d||G_\ell$ is depicted in Figure 2.14. Notice that there are three nontrivial SCCs, states* $(\{1N, 0Y\}, 1N)$,

$$\begin{tikzcd}
 & {} \arrow[d] & \\
 & (\{0N, 1Y\}, 0N) \arrow[dl, "a"'] \arrow[dr, "\sigma_f"] & \\
(\{2N, 3N, 2Y\}, 3N) \arrow[d, "b"'] & & (\{0N, 1Y\}, 1Y) \arrow[d, "a"] \\
(\{2N, 3N, 2Y\}, 2N) \arrow[d, "c"'] & & (\{2N, 3N, 2Y\}, 2Y) \arrow[d, "c"] \\
(\{2N, 2Y\}, 2N) \arrow[loop below, "c"] & & (\{2N, 2Y\}, 2Y) \arrow[loop below, "c"]
\end{tikzcd}$$

Figure 2.15: Automaton $G_{SCC} = G_d||G_\ell$ of Example 21

$(\{0Y, 1Y\}, 1N)$ *and* $(\{0Y, 1Y\}, 1Y)$. *Only state* $(\{0Y, 1Y\}, 1Y)$ *has a* Y*-labeled element of* G_ℓ*, however, component* $(\{0Y, 1Y\}$ *of* G_d *is not uncertain,i.e., contains elements* N*-labeled and* Y*-labeled. Since there does not exist a nontrivial SCC formed with states* (x_d, x_ℓ)*, such that* x_d *is uncertain and* x_ℓ *is a* Y*-labeled state, the language is diagnosable, as seen in Example 18.*

Example 21 *Consider automata* G_d *and* G_ℓ *of Example 19 depicted in Figure 2.13 (c) and (b), respectively. The new diagnoser automaton proposed* $G_{SCC} = G_d||G_\ell$ *is depicted in Figure 2.15. Notice that there are two nontrivial SCCs, states* $(\{2N, 2Y\}, 2N)$ *and* $(\{2N, 2Y\}, 2Y)$. *Only state* $(\{2N, 2Y\}, 2Y)$ *has a* Y*-labeled element of* G_ℓ*. Component* $(\{2N, 2Y\}$ *of* G_d *is uncertain. Since there exists a nontrivial SCC formed with states* (x_d, x_ℓ)*, such that* x_d *is uncertain and* x_ℓ *is a* Y*-labeled state, the language is not diagnosable, as seen in Example 19.*

Chapter 3

Hybrid systems

Hybrid systems (HS) are dynamic systems that combine the continuous dynamics with discrete event-based dynamics. Among many formalisms to describe a HS, such as *switched systems* [22], *condition-event systems* [23] or *labeled transition systems* [4], the *hybrid automaton* [4, 5, 24] stands out as one of the most relevant.

In this chapter we present some concepts on *hybrid systems* required for the development of this work. This chapter is organized as follows: in Section 3.1, we present a formal model for hybrid systems, called hybrid automata, denoted by H; in Section 3.2, we briefly present two approaches for diagnosability of HS, and, in Section 3.3, we present another formal modeling for HS, called transition systems, denoted by TS.

3.1 Hybrid automata

Hybrid automaton (HA) is a formal model for representing hybrid systems. It presents both event-driven discrete and continuous dynamics behavior. It is represented as a graph, where the vertices are the discrete states and the edges represent the transitions from one discrete state to another. The continuous dynamics is represented inside each vertice.

Definition 7 *A hybrid automaton is formally defined by a ten-tuple [5, 10, 19]:*

$$H = (\Sigma, Q, E, Q_0, X, f, I, G, R, X_0)$$

where:

- Σ *is the set of symbols or events;*
- Q *is the set of discrete states;*
- $E \subseteq Q \times \Sigma \times Q$ *is the transition relation;*

- *$Q_0 \subseteq Q$ is the set of initial discrete states;*
- *$X \subseteq \mathbb{R}^n$ is the continuous state space, where $n \in \mathbb{N}$;*
- *$f : Q \times X \to X$ is the vector field;*
- *$I : Q \to 2^X$ is the invariant;*
- *$G : E \to 2^X$ is the guard;*
- *$R : E \times X \to X$ is the reset function;*
- *$X_0 \subseteq X$ is the set of initial continuous states.*

The state of a HA is defined by the pair $(q, x) \in Q \times X$, where q is the discrete state, also called mode of operation or location, and x is the continuous state, both discrete and continuous time dynamical systems state.

The symbols in Σ are associated with events and the transitions trigger correspond to the occurrence of the events associated with them. For the location $q \in Q$, the vector field defines that the first derivative of the continuous state behaves as $\dot{x} = f(q, x(t))$ or $x(t+1) = f(q, x(t))$, and the invariant determines a condition of continuous state validity in the form $x \in I(q)$.

For the transition $e \in E$, the guard defines a condition for enabling the transition trigger from the continuous state in the form $x \in G(e)$. The reset sets a new value, x', when entering a new continuous state after the transition trigger according to $x' := R(e, x)$. To express that an event can occur independently of the condition in the continuous state we will make $G(e) = X$, *i.e.*, the guard condition is satisfied for the entire continuous state space.

3.1.1 Hybrid solution

A *solution* to a HA is a pair of right-hand continuous signals $x : [0, \infty) \to X$ and $q : [0, \infty) \to Q$, called trajectories, as studied in ALUR *et al.* [13]:

- $x(t)$ is piecewise differentiable and $q(t)$ is piecewise continuous;
- In any interval (t_1, t_2) where $q(t)$ is constant and $x(t)$ is continuous and defined for all $t \in [t_1, t_2)$ as:

  $$x(t) \quad = \quad \phi(q(t_1), x(t_1), t)$$

  where $\phi(q, x(t_0), t)$ is the solution of $\dot{x} = f(q, x)$ for $t \geq t_0$ with initial condition $x(t_0)$; and

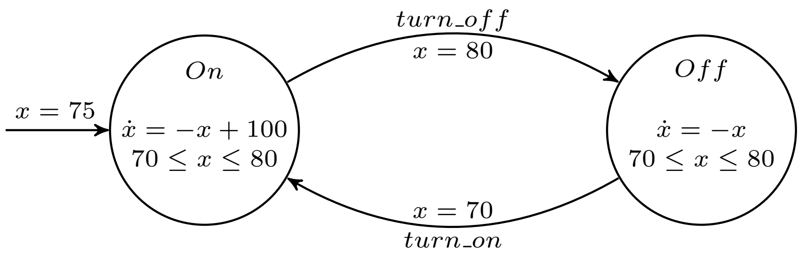


Figure 3.1: Hybrid automaton H_1 of a thermostat of Example 22

- For any instant $t \geq 0$, $(q(t), x(t))$ is such that there exists $e = (q_1, \sigma, q_2)$ with $q(t^-) = q_1$, $q(t^+) = q_2$, $x(t^-) \in G(e)$ and $x(t^+) := R(e, x(t^-))$.

In the following examples we will show the difference of continuous trajectories for an automaton whose transition occurs as soon as the guard condition is satisfied and when it occurs any time between the guard condition and the invariant limit.

Example 22 *Let us consider the hybrid automaton H_1 represented in Figure 3.1, which models a thermostat. For this automaton we can identify the following elements: the two locations $Q = \{On, Off\}$; the initial location $Q_0 = \{On\}$; the set of events $\Sigma = \{turn\_on, turn\_off\}$; the transitions $On \xrightarrow{turn\_off} Off$ and $Off \xrightarrow{turn\_on} On$; the continuous state is a single element vector, $x = [x]^T$, which represents the temperature and the initial continuous state $X_0 = \{75\}$. The continuous dynamics is $\dot{x} = -x + 100$ the location On and $\dot{x} = -x$ the location Off. The invariants are $70 \leq x \leq 80$ for both locations. Note that the limits of the invariants are exactly the guard conditions, i.e., the transition must occur as soon as the guard is satisfied. Also note that omitting the restart condition of a transition means that it corresponds to the identity, which is the case of the transitions in this example. The continuous states enter the new location with the exact same value as they had when the transition was triggered.*

The locations in Figure 3.1 are drawn as circles with the location name on top and all transitions are drawn as arcs connecting the source location q to $q \xrightarrow{\sigma}$ labeled by event σ.

Notice that, for the Example 22, we can obtain both continuous and discrete (location) trajectories, as seen in Figure 3.2. The former is represented in the $Temperature \times Time$ graph, whereas the latter is represented in the $State \times Time$ graph, where locations On and Off are represented by the numbers 1 and 0, respectively. We can note that the transitions always occur when the current location is On and $x = 80$ and when the current location is Off and $x = 70$.

Example 23 *Let us consider the hybrid automaton H_2 represented in Figure 3.3, which also models a thermostat. It has the same configuration as the previous HA H_1*

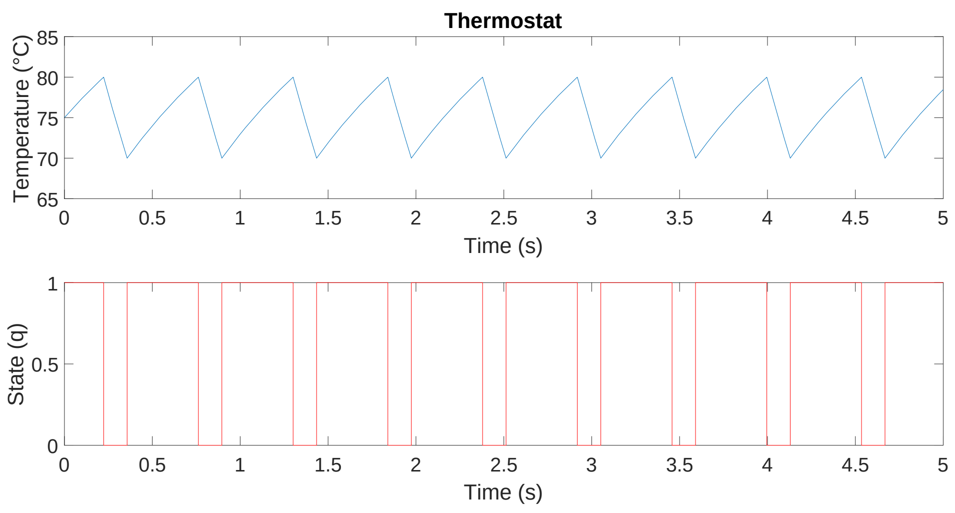


Figure 3.2: Continuous and discrete (location) trajectories for the thermostat example 22

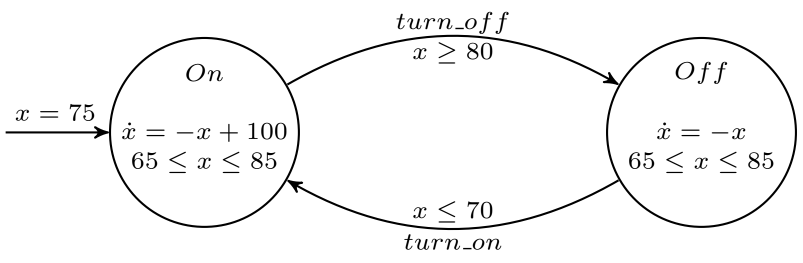


Figure 3.3: Hybrid automaton H_2 of a thermostat of Example 23

apart from the guards and invariants. For this HA, the invariants are $65 \leq x \leq 85$ *for both locations, i.e., it can transition any time between the guard condition is satisfied and the limit of the invariant for the current location. If the current location is* On*, it can transition when the continuous state assume any value in the interval* $[80, 85]$*. And when* Off *is the current location, it can transition any time in the interval* $[65, 70]$ *for the continuous state values.*

We can easily confirm it in Figure 3.4, where it shows the continuous and discrete (location) trajectories for the example 23. The first one is represented in the $Temperature \times Time$ graph and the last one is represented in the $State \times Time$ graph, where locations On and Off are represented by the numbers 1 and 0, respectively. Whenever the current state is On, the transitions occur when the continuous state values are greater than 80 and less than 85. And whenever the current state is Off, the transitions occur when the continuous state values are greater than 65 and less than 70.

In this work we will work with finite switching systems, where there is a time interval between each switching. The calculation of the hybrid solution is fundamental

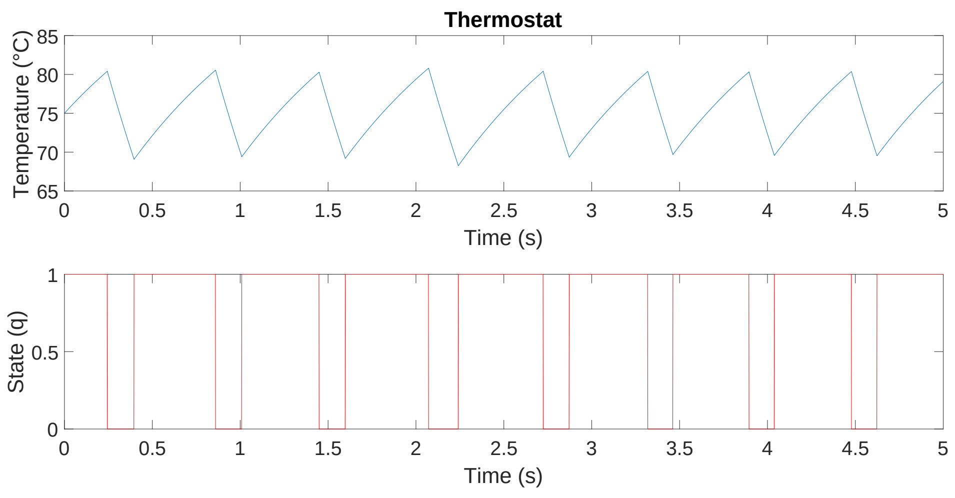


Figure 3.4: Continuous and location trajectories for the thermostat example 23

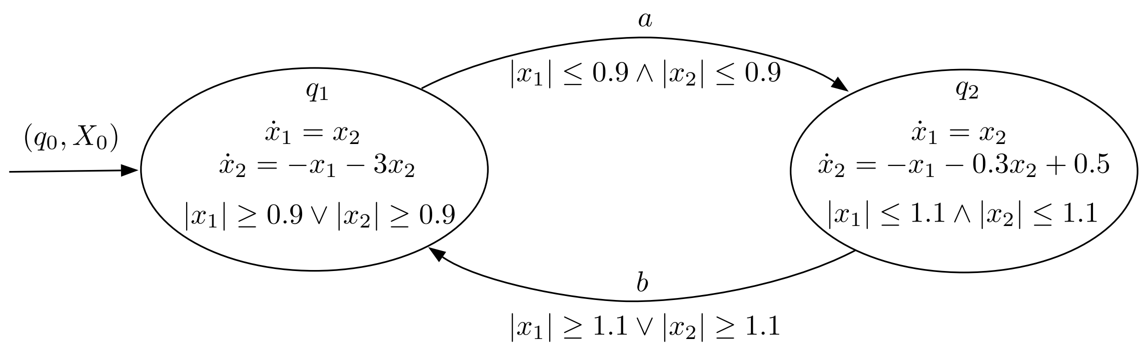
to obtain the reachable region of the system. The determination of the reachable region for hybrid systems is a more challenging problem than for the continuous systems due to the influence of the events in the continuous evolution.

3.1.2 Reachable region

The reachable region $\mathcal{R}_H(Q_i, X_i)$ is a set of reachable states. It is calculated by an interactive algorithm which successively obtains the successor states sets of a certain set of states by discrete transitions and by continuous evolution [13]. The idea is to calculate the set of reachable states by a small neighborhood of a finite set of trajectories to form the reachable region. For this purpose, computational tools compute $\mathcal{R}_H(Q_i, X_i)$ employing symbolic states (q, Y), where $q \in Q$ and $Y \subseteq X$ is a continuous set represented by a geometric entity, such as a polyhedron, an ellipsoid, a zonotope, or a support function, among others kinds of approximations [14–16].

Example 24 details a deterministic hybrid automaton model. In the deterministic HA, the transition function is unique for each location and guard, *i.e.*, there is no more than one transition from the same location reaching different locations subject to the same guard.

Example 24 *Let us consider the hybrid automaton H represented in Figure 3.5. For this automaton we can identify the following elements: the two locations $Q = \{q_1, q_2\}$; the set of events $\Sigma = \{a, b\}$; the transitions $q_1 \xrightarrow{a} q_2$ and $q_2 \xrightarrow{b} q_1$; the continuous states, represented in a vector form, $x = [x_1\ x_2]^T$. The continuous dynamics is $\dot{x}_1 = x_2$ and $\dot{x}_2 = -x_1 - 3x_2$ in location q_1 and $\dot{x}_1 = x_2$ and $\dot{x}_2 = -x_1 - 0.3x_2 + 0.5$ in location q_2. The invariants are $|x_1| >= 0.9 \vee |x_2| >= 0.9$ in location q_1, which represents the exterior of a square of side 1.8 centered on the*



a
$|x_1| \leq 0.9 \wedge |x_2| \leq 0.9$
(q_0, X_0)
q_1
$\dot{x}_1 = x_2$
$\dot{x}_2 = -x_1 - 3x_2$
$|x_1| \geq 0.9 \vee |x_2| \geq 0.9$
q_2
$\dot{x}_1 = x_2$
$\dot{x}_2 = -x_1 - 0.3x_2 + 0.5$
$|x_1| \leq 1.1 \wedge |x_2| \leq 1.1$
b
$|x_1| \geq 1.1 \vee |x_2| \geq 1.1$

Figure 3.5: Hybrid automaton H of Example 24.

origin of $x_1 \times x_2$, and $|x_1| <= 1.1 \wedge |x_2| <= 1.1$ in location q_2, which represents the interior of a square of side 2.2, also centered at the origin of $x_1 \times x_2$. Notice that for the transition labeled by event a, the guard condition is that the state reaches the perimeter of the square of side 1.8, while, for the transition labeled by event b, the guard condition corresponds to the perimeter of the square of side 2.2. Also note that omitting the restart condition of a transition means that it corresponds to the identity, which is the case of the transitions in this example.

A proper set approximation is very important to reachability analysis. Set operations may lead to an enormous error accumulation and even an unsolvable situation. For this purpose, set representation has been studied and improved, especially convex set representations. In words, given any points x_1, x_2 belonging to a convex set C, the line segment between them is contained in C [25]. The next subsections aim to provide some basic definitions of the most common convex set representations.

Polyhedra

A polyhedron is an intersection of finitely many halfspaces. Polyhedra are convex sets of the form $P = \{x|Ax \leq b, Cx = d\}$. A polytope is a bounded polyhedron.

Template polyhedra

A template polyhedron is a polyhedron with faces whose normal vectors are given a priori. Template polyhedra are sets of the form $P_D = \{x \in \mathbb{R}^n | \bigwedge_{\ell_i \in D} \ell_i \cdot x \leq b_i\}$, where $D = \{\ell_1, \cdots, \ell_m\}$ is the set of template directions $\ell_i \in \mathbb{R}^n$.

Ellipsoid

Ellipsoids ε can be represented as $\varepsilon(c, Q) = \{x : (x - c) \cdot Q^{-1}(x - c) \leq 1\}$, where c is the center of the ellipsoid and Q its positive define shape matrix.

Zonotope

Zonotopes are a special type of polytopes, such that $Z = \{x \in \mathbb{R}^n | x = c + \sum_{i=1}^{p} \alpha_i g_i, -1 \leq \alpha_i \leq 1\}$, where c is the center and g_i are called the generators of Z, $c, g_i \in \mathbb{R}^n$.

Support functions

Support Function ρ_s of a set S is defined by

$$\begin{aligned} \rho_s : \mathbb{R}^d &\rightarrow \mathbb{R} \cup \{-\infty, \infty\} \\ \ell &\mapsto \sup_{x \in S} x \cdot \ell, \end{aligned}$$

where ℓ is a direction vector and ρ_s is the solution to the maximization of $x \cdot \ell$ for $x \in S$. Support functions represent convex sets by a function instead of a set of parameters as the previous representations. The ρ_s indicates where the hyperplane orthogonal to ℓ must be placed. So for each vector ℓ we are able to define a hyperplane orthogonal to ℓ which touches S at one point.

In general, we work with a conservative approximation $\tilde{\mathcal{R}}_H(Q_i, X_i)$ of a reachable region $\mathcal{R}_H(Q_i, X_i)$, in the sense that calculations guarantee that $\tilde{\mathcal{R}}_H(Q_i, X_i) \supseteq \mathcal{R}_H(Q_i, X_i)$ [14].

Figure 3.6 shows the reachable region of the system describe in Example 24 for the initial condition $(q_1, [[1.9, 2.1]\ 0]^T)$. The graph was obtained with the help of *SpaceEx* tool [16, 17]. In this specific case, we can observe that the reachable region is approximated by *support functions* [15]. Briefly, support functions are a form of representing any set S by a function which takes into consideration a direction ℓ and the maximum product $x \cdot \ell, x \in S$ in order to place a hyperplane orthogonal to ℓ touching S.

Hybrid systems have properties that will be properly discussed in Subsection 3.3.2, that frequently have to be verified in order to ensure the proper behavior of the system. The verification of properties of hybrid systems employs the *reachability analysis* to compare a given reachable region from a set of initial states, $\mathcal{R}_H(Q_i, X_i)$, with a set $F \subseteq Q \times X$ which characterizes a desired property [14].

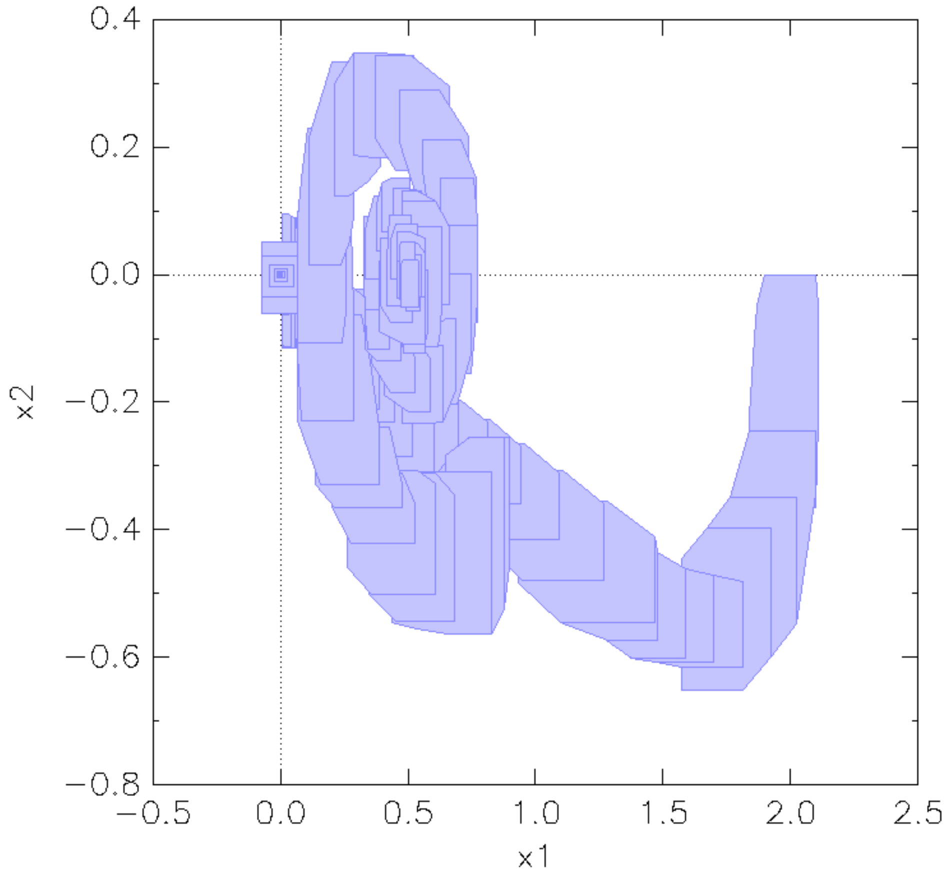


Figure 3.6: Example of the reachable region for the hybrid automaton of Example 24.

3.2 Diagnosability of hybrid systems

This section is a literature review of works [1, 10] where two methods for diagnosablitity of hybrid systems are presented. In section 3.2.1, an event-driven diagnoser automaton based on the abstraction of the continuous dynamics is proposed, and, in section 3.2.2, an event-driven verifier automaton which implements the distinguishability of the continuous states is presented.

3.2.1 Diagnosability analysis of hybrid systems cast in a discrete-event framework

This section is based on BAYOUDH and TRAVÉ-MASSUYÈS [10] work, where it is adopted the concept of discretization of the continuous time-driven dynamics by introducing a new concept of mode signature. A behavior automaton is constructed by enriching the underlying DES with new events created from the detectability of mode signature changes. DES fault diagnosis techniques are adopted on the behavior automaton to determine diagnosability. Let a hybrid system be modeled by the hybrid automaton $H = (\xi, Q, \Sigma, \phi, C, (q_0, \xi_0))$, where Q and Σ are defined as in Section 3.1, ξ is the set of continuous variables, as continuous state, input, output and noise, $\phi \subseteq Q \times \Sigma \to Q$ is the partial transition function, C is the set of system constraints and $(q_0, \xi_0) \in Q \times \xi$ is the initial condition of the HS. Notice that some notations have been modified from Section 3.1 for being slightly different or for not being defined previously.

The system behavior captured through event-based dynamics is called underlying DES model represented by the automaton $G = (Q, \Sigma, \phi, q_0)$ and the system behavior captured through continuous time-driven dynamics is called multimode system represented by $\Theta = (\xi, Q, C, \xi_0)$.

Let the signature of a mode q_i be denoted as $Sig(q_i)$ be $Sig(q_i) = [S^T_{i/1}, S^T_{i/2}, \cdots, S^T_{i/i}, \cdots, S^T_{i/n_r}]$, where n_r is the number of modes of the behavior automaton, $S^T_{i/j}$ is the vector of size n_i, number of residuals, formed by booleans. Whenever a residual of q_i is consistent with a residual of q_j, it is assumed a 0 value, 1 otherwise. The signature of a mode represents the expected behavior of a particular mode with respect to all other modes.

Definition 8 *Two modes q_i and q_j are diagnosable if $Sig(q_i) \neq Sig(q_j)$. A multimode system Θ is mode diagnosable if and only if all pairs of modes q_i and q_j, $i \neq j$, are diagnosable.*

Let Σ^{Sig} be a set of discrete events, called *signature events*, associated with mode signature change detection. Whenever a mode q_i transitions to a mode q_j, $i \neq j$, an event δ_{ij} is created, where $\delta_{ij} = R_{O_{ij}}$ is observable if $Sig(q_i) \neq Sig(q_j)$ and $\delta_{ij} = R_{UO_{ij}}$ is unobservable, otherwise. Let Q_t be the set of transient modes, which are modes added to the underlying DES associated with mode signature change, $q_{ij} \in Q_t$ is a transient mode placed between modes q_i and q_j.

The analytical redundancy relation used to calculate residual consistency in BAYOUDH and TRAVÉ-MASSUYÈS [10] work is based on the parity space approach. It is assumed that the discrete dynamics is slower, in order of magnitude, than the residual consistency.

Let the automaton generating language L_A be the behavior automaton defined by $B_A = (\bar{Q}, \bar{\Sigma}, \bar{\phi}, q_0)$, where $\bar{Q} = Q \cup Q_t$, $\bar{\Sigma} = \Sigma \cup \Sigma^{Sig}$, $\bar{\phi} = \bar{\phi}_1 \cup \bar{\phi}_2$. with $\bar{\phi}_1 \subseteq (Q \times \Sigma \to Q_t)$ and $\bar{\phi}_2 \subseteq (Q_t \times \Sigma^{Sig} \to Q)$, *i.e.*, every transition of the underlying DES is replaced by two transitions, the first from mode q_i to the transient mode q_{ij} with event from the underlying DES and then from transition mode q_{ij} to mode q_j with a signature event.

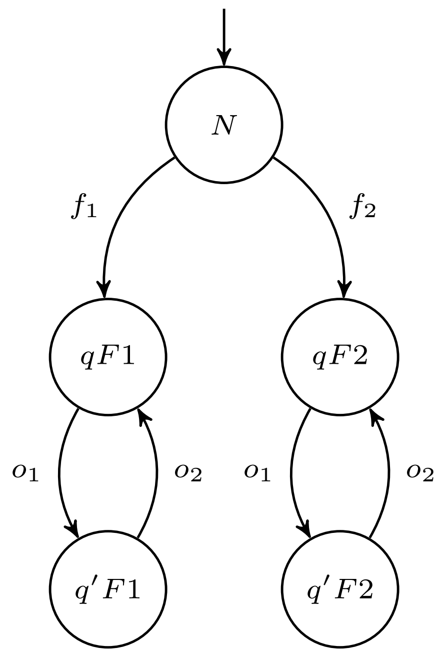


Figure 3.7: Underlying DES automaton G of Example 25

Definition 9 *The hybrid system is diagnosable if $\forall \sigma_f \in \Sigma_f$, $\exists n \in \mathbb{N}$ such that $\forall s_F t \in L_A$, where s_F is a trace ending with a fault event σ_f, and $t \in L_A$ is a continuation of s_F:*

$$|t| \geq n \implies (\forall \omega \in L_A : P_{\bar{\Sigma}_o(\omega)} = P_{\bar{\Sigma}_o(s_F t)} \implies \sigma_f \in \omega)$$

where $P_{\bar{\Sigma}_o}$ is the projection operator on the set $\bar{\Sigma}_o = \Sigma_o \cup \Sigma_o^{Sig}$.

In words, a hybrid system is diagnosable if for every sequence with the same projection as a faulty sequence is necessarily a faulty sequence. The diagnoser automaton is constructed from the behavior automaton applying DES tools presented in Section 2.6 and is the tool adopted to analyze diagnosability.

Example 25 *Let G be the underlying DES extracted from BAYOUDH and TRAVÉ-MASSUYÈS [10] represented in Figure 3.7 with the set of states $Q = \{N, qF1, qF2, q'F1, q'F2\}$, where N is a normal mode and $qF1$ and $qF2$ are two faulty modes reached after the occurrence of the fault events f_1 and f_2, respectively. Let the set of observable events $\Sigma_o = \{o_1, o_2\}$ and the set of fault events $\Sigma_f = \{f_1, f_2\} \subseteq \Sigma_{uo}$. Consider the continuous dynamics of each mode given by the state-space model below:*

$$\begin{cases} \dot{x}(t) = A_i x(t) + B_i u(t) \\ y(t) = C_i x(t) + D_i u(t) \end{cases}$$

where $i = 1, 2, \cdots, 5$ corresponds to modes N, $qF1$, $qF2$, $q'F1$ and $q'F2$, respec-

tively.

$$A_1 = \begin{bmatrix} -1 & 1 \\ 0 & -1 \end{bmatrix}, \; A_2 = \begin{bmatrix} -2 & 1 \\ 0 & -2 \end{bmatrix}, \; A_3 = \begin{bmatrix} -\frac{8}{3} & \frac{4}{3} \\ -\frac{1}{3} & -\frac{4}{3} \end{bmatrix},$$

$$A_4 = \begin{bmatrix} -3 & 1 \\ 0 & -3 \end{bmatrix}, \; A_5 = \begin{bmatrix} -4 & 1 \\ 0 & -4 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \; B_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \; B_3 = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \; B_4 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \; B_5 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 1 & 1 \end{bmatrix}, \; C_2 = \begin{bmatrix} 0 & 1 \end{bmatrix}, \; C_3 = \begin{bmatrix} -\frac{1}{3} & \frac{2}{3} \end{bmatrix}, \; C_4 = \begin{bmatrix} 1 & 0 \end{bmatrix}, \; C_5 = \begin{bmatrix} 2 & 2 \end{bmatrix}$$

$$D_1 = D_2 = D_3 = D_4 = D_5 = 1$$

Modes $qF1$ and $qF2$ have the same input/output behavior, thus their residuals are the same and taken one time in the mode signatures, $Sig(q_i) = [S^T_{i/N}, S^T_{i/qF1} = S^T_{i/qF2}, S^T_{i/q'F1}, S^T_{i/q'F2}]^T \in \mathbb{R}^5$. The signature vectors are represented as follows:

$$Sig(N) = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \; Sig(qF1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \; Sig(qF2) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix},$$

$$Sig(q'F1) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{bmatrix}, \; Sig(q'F2) = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

The abstraction of the underlying system G is given by the behavior automaton B_A represented in Figure 3.8. Notice that all of its signature-events are observable since the mode signature of the source mode is different from the mode signature of the destination mode for all the modes. The extended set of events is $\bar{\Sigma} = \{o_1, o_2, f_1, f_2, Ro_{12}, Ro_{23}, Ro_{24}, Ro_{32}, Ro_{42}\}$

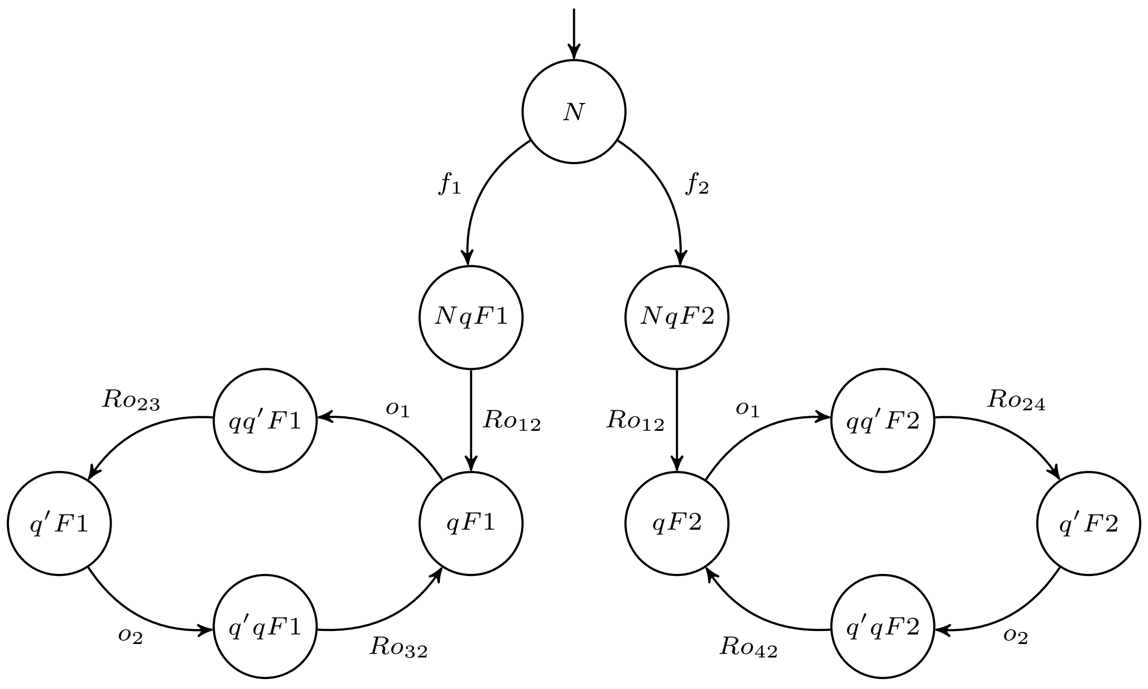


Figure 3.8: Behavior automaton B_A for the automaton G of Example 25

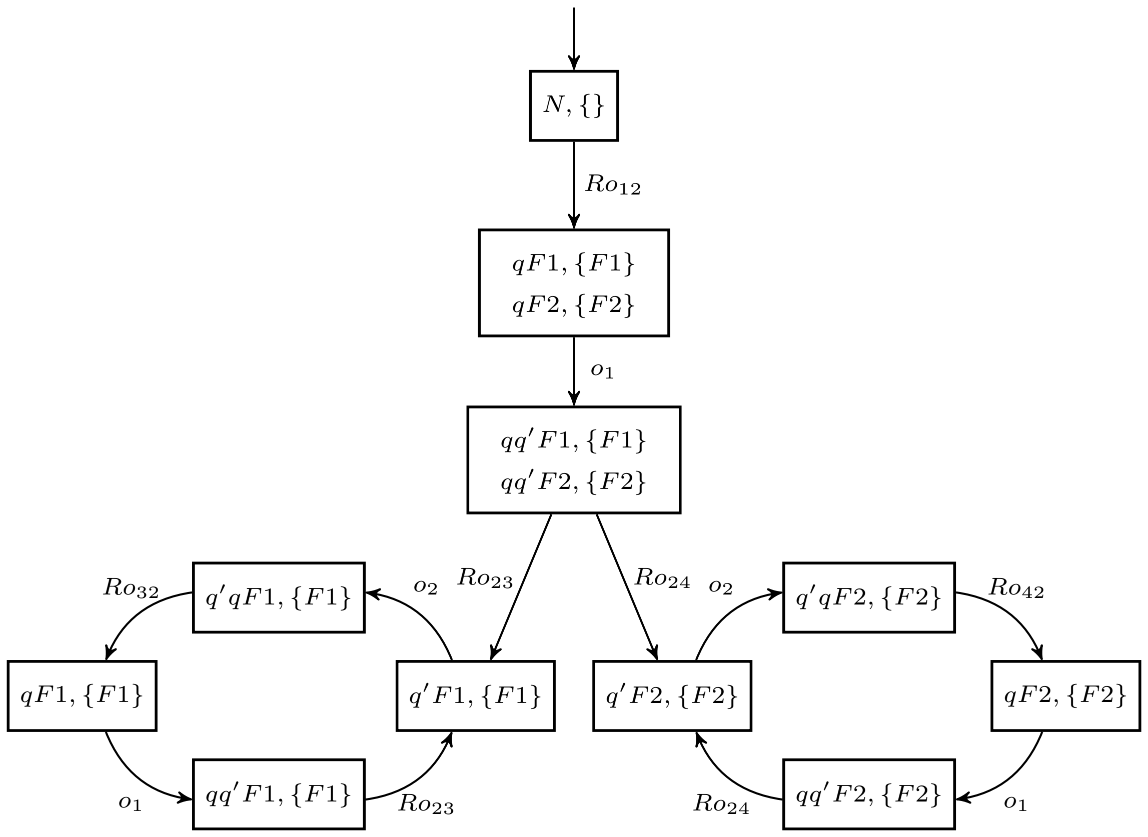


Figure 3.9: Diagnoser automaton $Diag(B_A)$ for the behavior automaton B_A of Example 25

The diagnoser automaton $Diag(B_A)$ of the behavior automaton is displayed in Figure 3.9. It is constructed as in Section 2.6, where the label automaton distinguish the fault types and gives an empty label when the state has not been reached by a fault event. Since there are no cycles with a normal and a fault label and there are no cycles with more than one fault label, the system is diagnosable.

3.2.2 Verification of diagnosability of hybrid systems

This section is based on DIENE *et al.* [1] work, where they introduce a new concept of diagnosability called h-diagnosability and present a verification method for this property based on the DES verifier automaton and the distinguishability of continuous modes. Let the deterministic hybrid system denoted as $H = (Q, X, Y, Z, \Sigma, U, f_u, g, \phi, \Gamma, R, q_0, x_0)$, where Q, X, Σ and R are defined as in Section 3.1, $Y \subseteq \mathbb{R}^p$ is the set of outputs, $Z \subseteq \mathbb{R}^r$ is the set of noises, $U \subseteq \mathbb{R}^m$ is the set of controls, $f_u : Q \times X \times U \times Z \to X$ is the vector field, $g : Q \times X \times U \times Z \to Y$ is the output function, $\phi : Q \times \Sigma \to Q$ is the discrete state transition function, $\Gamma : Q \to 2^\Sigma$ is the feasible event function, $q_0 \in Q$ is the initial discrete state and $x_0 \in X$ is the initial continuous state. Notice that some notations have been modified from [1] to better fit to the nomenclature adopted in this work.

Without loss of generality, all transitions triggered by continuous dynamics are modeled as events. The system behavior captured through event-based dynamics is called underlying DES model represented by the deterministic automaton $G = (Q, \Sigma, \phi, \Gamma, q_0)$. The continuous dynamics are assumed as linear and time invariant.

Let the vector field $f_u(q_i, x, u, \xi)$ and the output function $g(q_i, x, u, \xi)$ associated with each mode q_i be given by the continuous sate model denoted as $\mathcal{M}_i$ described in the following:

$$\begin{cases} x(n+1) = A_i x(n) + B_i u(n) + E_{x_i} \xi(n) \\ y(n) = C_i x(n) + D_i u(n) + E_{y_i} \xi(n) \end{cases}$$

where T_s is the sampling period, $x(n)$, $u(n)$, $y(n)$ and $\xi(n)$ are the state, input, output and noise vectors at time nT_s and A_i, B_i, C_i, D_i, E_{x_i} and E_{y_i} are constant matrices.

Definition 10 *Let q_i and q_j be different modes of H, whose associated continuous state models are $\mathcal{M}_i$ and $\mathcal{M}_j$, respectively. The modes q_i and q_j are distinguishable ($q_i \nsim q_j$) with respect to a continuous state ψ and a tolerance μ, if $\psi(\mathcal{M}_i, \mathcal{M}_j, u, \xi, x_0') > \mu$, where u, ξ, and x_0' denote the input and the noise defined over a time interval I, and the initial condition of both models, respectively. The modes q_i and q_j are undistinguishable ($q_i \sim q_j$), otherwise.*

Let the unobservable reach of a mode $q_i \in Q$ with respect to the set of unobservable events Σ_{uo}, denoted by $UR(q_i)$ be defined as

$$UR(q_i) = \{q_j \in Q : (\exists t \in \Sigma_{uo}^*)[\phi(q_i, t) = q_j]\}$$

And can be extended to a subset of states $\Theta \subseteq Q$

$$UR(\Theta) = \bigcup_{q_i \in \Theta} UR(q_i)$$

Let $Reach(G, \nu)$ be the estimate current states after the execution of a trace $s \in L$, with projection $P_o(s) = v\sigma_o = \nu$ ending with an observable event $\sigma_o \in \Sigma_o$

$$\begin{aligned} Reach(G, \varepsilon) &= UR(q_0) \\ Reach(G, v\sigma_o) &= UR(\triangle(Reach(G, v), \sigma_o)) \end{aligned}$$

where $\triangle(Reach(G, v), \sigma_o) = \bigcup_{i=1}^{k} \delta(q_i, \sigma_o)$, with $q_i \in Reach(G, v)$, $k = |Reach(G, v)|$, and $\delta(q_i, \sigma_o) = \{\phi(q_i, \sigma_o)\}$, if $\phi(q_i, \sigma_o)$ is defined and $\delta(q_i, \sigma_o) = \emptyset$, otherwise.

In the distinguishability analysis between two continuous models $\mathcal{M}_i$ and $\mathcal{M}_j$, the input u and initial state x'_0 are assumed to be the same for both models. The distinguishability analysis adopted in DIENE *et al.* [1] work is the parity space approach. It is assumed that no event can occur during this analysis.

Definition 11 *Let L denote the language generated by the underlying DES model G of an HS and $L_N \subseteq L$ be the normal language of G. The HS modeled by H is said to be h-diagnosable if*

$$(\exists n \in \mathbb{N})(\forall s \in L \setminus L_N)(\forall st \in L \setminus L_N, |t| \geq n) \implies D_1 \vee D_2$$

where D_1:

$$(\forall \omega \in P_o^{-1}(P_o(st)) \cap L, \omega \in L \setminus L_N)$$

and D_2:

$$(\forall s_N \in L_N, P_o(s_N) = P_o(st) = \nu)(\exists \eta\sigma_o \in \overline{\nu}, q_F \nsim q_N)$$

where $q_F \in \triangle(Reach(G_{\mathcal{P}_F}, \eta), \sigma_o)$ (resp. $q_N \in \triangle(Reach(G_{\mathcal{P}_N}, \eta), \sigma_o)$) and $G_{\mathcal{P}_F}$ (resp. $G_{\mathcal{P}_N}$) is the subautomaton of G formed with path $\mathcal{P}_F$ (resp. $\mathcal{P}_N$) with corresponding trace st (resp. s_N).

In words, after the occurrence of a fault event, the HS H is diagnosable if at least one of the two conditions are satisfied. One way is if the projections of a normal and a faulty trace are different from each other (condition D_1) and another way is the distinguishability of the modes reached after the occurrence of an observable event.

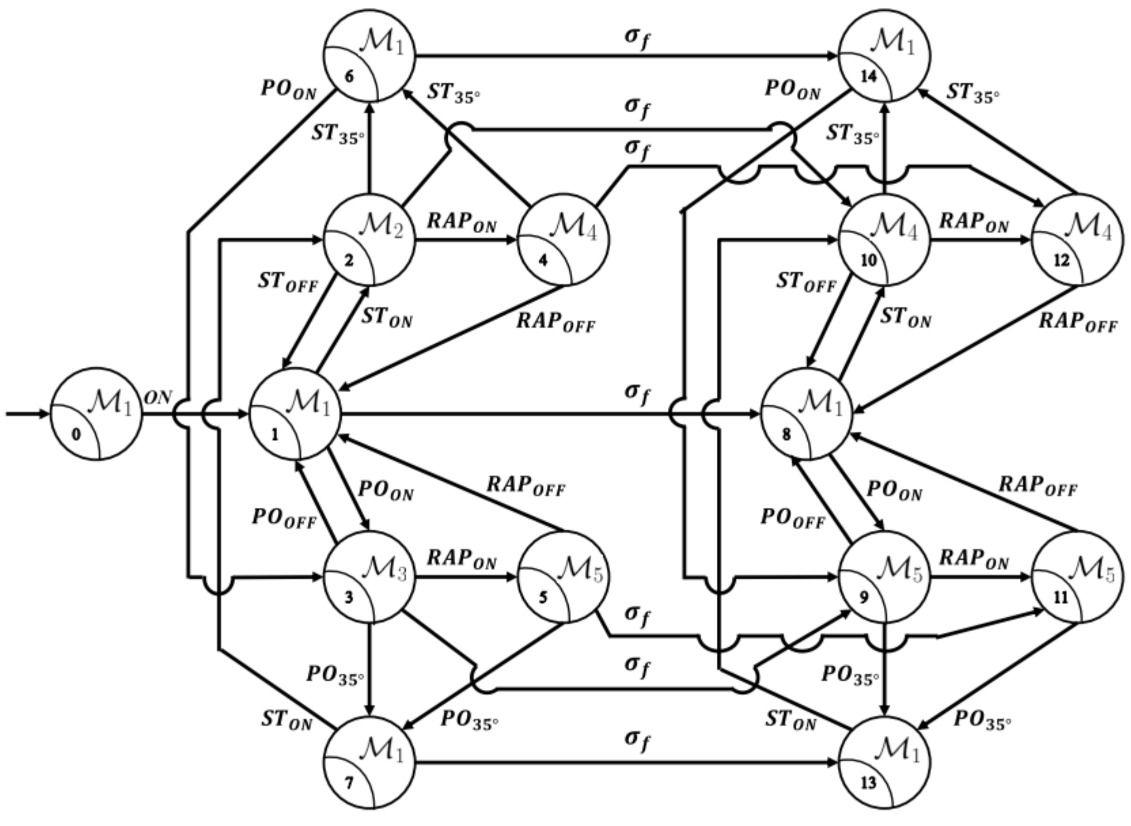


Figure 3.10: Underlying DES automaton G of Example 26. Figure extracted from [1]

The method proposed in DIENE *et al.* [1] consists of building two automata G_N^{HT} and G_F^{HT} which represent the normal and fault behaviors of the underlying DES model labeled with H for states reached after the occurrence of an observable event and T for states reached after the occurrence of an unobservable event. The unobservable events of G_N^{HT} are renamed becoming private events of the new automaton $G_{N_R}^{HT}$. Finally the verifier automaton G_{VH} is computed by applying the *Paralleld* function with $G_{N_R}^{HT}$ and G_F^{HT} as its arguments, where it performs a modified parallel function. The distinguishability method is applied and whenever two states labeled with H are reached and distinguished, the path is interrupted. The HS is h-diagnosable if there is no cyclic path in the constructed automaton G_{VH} where a state in the cyclic has label Y obtained from automaton G_F^{HT} and the event that reaches this state is in the set of events of the automaton G.

Example 26 *Let G be the underlying DES extracted from DIENE* et al. *[1] be the model of a rudder of a ship depicted in Figure 3.10. Let the set of observable events* $\Sigma_o = \{ON, PO_{ON}, PO_{OFF}, ST_{ON}, ST_{OFF}, RAP_{ON}, RAP_{OFF}, PO_{35°}, ST_{35°}\}$ *and the set of fault events* $\Sigma_f = \{\sigma_f\} \subseteq \Sigma_{uo}$*. The state variables* $x_1(n)$*,* $x_2(n)$ *and* $x_3(n)$ *represent the angular position, speed and acceleration of the rudder. Let the sampling*

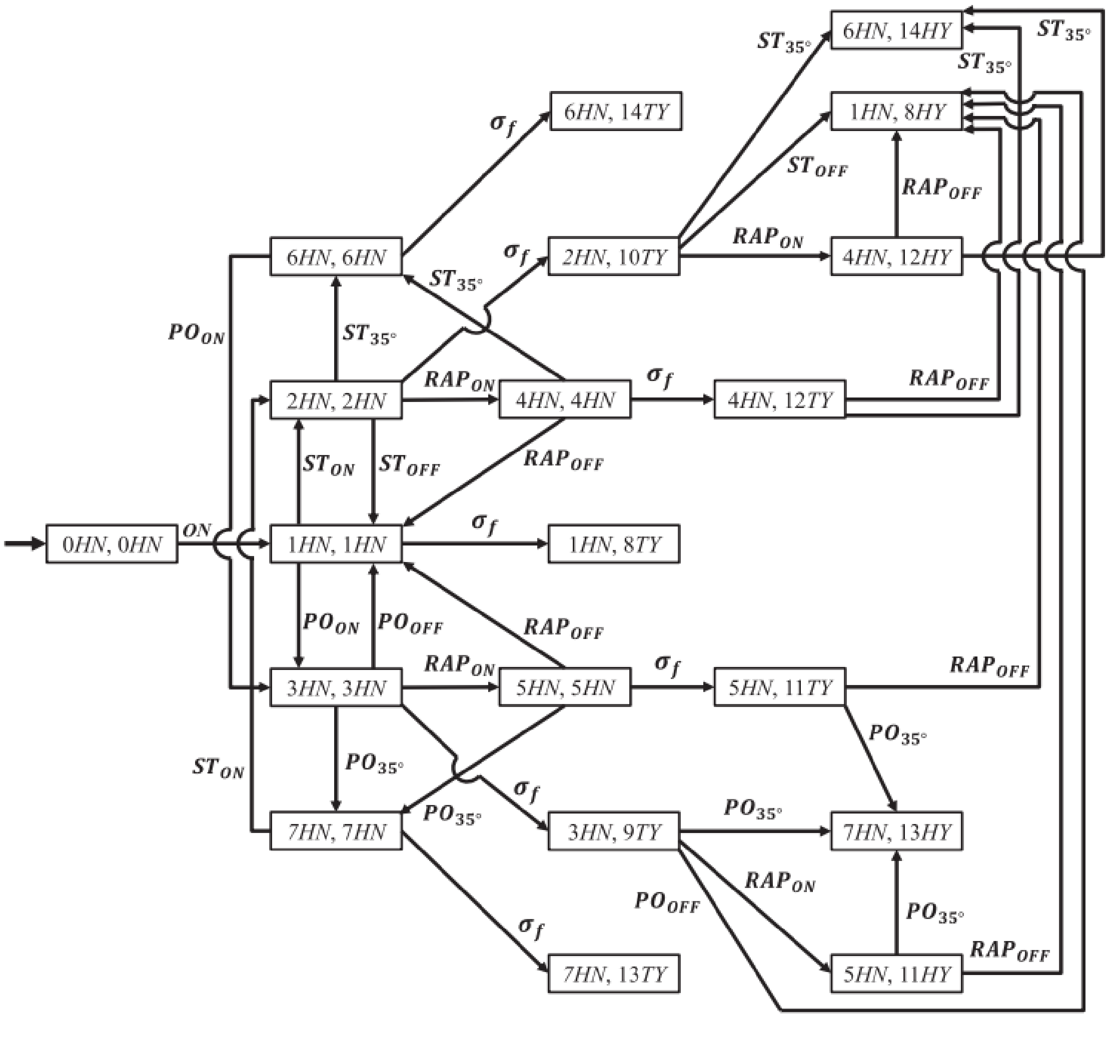


Figure 3.11: Verifier automaton G_{VH} of Example 26. Figure extracted from [1]

period $T_s = 0.001s$ and the continuous model constant matrices given as follows:

$$A_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \; A_2 = A_3 = \begin{bmatrix} 1 & 0.001 & 0 \\ 0 & 0.9997 & 0.0009 \\ -0.0462 & -0.596 & 0.7665 \end{bmatrix},$$

$$A_4 = A_5 = \begin{bmatrix} 1 & 0.001 & 0 \\ 0 & 0.9996 & 0.0009 \\ -0.0447 & -0.7176 & 0.7176 \end{bmatrix}$$

$$B_1 = 0, \; B_2 = \begin{bmatrix} 0 \\ 0.0017 \\ 3.2725 \end{bmatrix}, \; B_3 = -B_2, \; B_4 = \begin{bmatrix} 0 \\ 0.0026 \\ 4.9558 \end{bmatrix}, \; B_5 = -B_4$$

$$C_1 = C_2 = C_3 = C_4 = C_5 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

$$D_1 = D_2 = D_3 = D_4 = D_5 = 0$$

Modes 3 and 9 and modes 2 and 10 are distinguishable, i.e., $3 \nsim 9$ and $2 \nsim 10$. The pair of modes $1 \sim 8$, $4 \sim 12$, $5 \sim 11$, $6 \sim 14$ and $7 \sim 13$ are undistinguishable. The verificator automaton G_{VH} is depicted on Figure 3.11. We can verify that there is no cyclic path in G_{VH} with both labels N and Y, therefore, the hybrid system is h-diagnosable.

3.3 Transition systems

Transition systems (TS) are another form of representation of HSs behavior. They are represented by graphs, where their vertices represent the states and the edges model transitions, *i.e.*, state changes. [4, 5, 26]. They are often used for checking system properties, also known as model checking. This section is based on BAIER and KATOEN [26] and CLARKE *et al.* [27].

Definition 12 *A transition system can be described as a tuple*

$$TS = (S, Act, \rightarrow, I, AP, L)$$

where:

- *S is the set of states;*
- *Act is the set of actions;*
- *$\rightarrow \subseteq S \times Act \times S$ is the transition relation;*

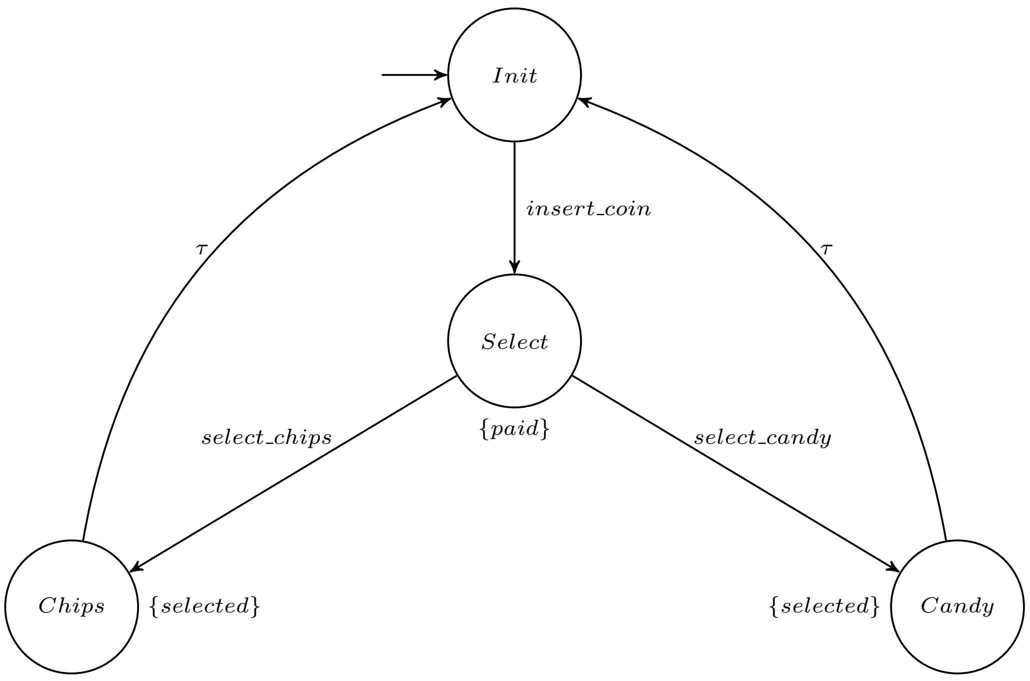


Figure 3.12: Snack vending machine transition system of Example 27.

- *$I \subseteq S$ is the set of initial states;*
- *AP is the set of atomic propositions; and*
- *$L : S \to 2^{AP}$ is the labeling function.*

A state carries information about the system at a given moment of its behavior and the transitions determine how the system can evolve. When there is more than one initial state, the start state is chosen in a nondeterministic way. Actions are transition labels that can be used to describe communication mechanisms. The states are labeled with atomic propositions related by the labeling function. The atomic propositions can be understood as the expression of facts about the states.

We will represent the occurrence of a transition from a state s to s' labeled with action α by $s \xrightarrow{\alpha} s'$. While in DES the focus is on the event sequences, in TS the focus is in state sequences and their labeling function values.

Example 27 *Let us consider the transition system TS represented in Figure 3.12 which models a snack vending machine. For this system we can identify the following elements: there are four states $S = \{Init, Select, Chips, Candy\}$; the set of actions $Act = \{insert\_coin, select\_candy, select\_chip, \tau\}$; the transitions $\rightarrow = \{Init \xrightarrow{insert\_coin} Select, Select \xrightarrow{select\_candy} Candy, Select \xrightarrow{select\_chips} Chips, Candy \xrightarrow{\tau} Init, Chips \xrightarrow{\tau} Init\}$; the initial state $I = \{Init\}$; the atomic propositions $AP = \{paid, selected\}$ and the labeling functions $L(Init) = \emptyset$, $L(Select) = paid$ and $L(Chips) = L(Candy) = selected$.*

At the initial state, when the user inserts a coin, the system goes to state Select. The atomic proposition of this state means that the user has already paid for the snack. The user's next action is to choose a snack, and the next states have the selected atomic proposition. In order to get back to its initial state, the system must have received the money and have a snack selected. Note that the action τ does not have any meaning, it could be compared to the ε event in DES. After the snack is selected the system delivers the snack and goes back to its start state.

The *direct successor operator (Post)* and *direct predecessor operator (Pre)* return the set of direct successors and predecessors, respectively. They are used in reachability analysis.

Definition 13 *Let TS be a transition system $s \in S$ and $\alpha \in Act$, the set of direct $\alpha - successors$ of s is expressed as:*

$$Post(s, \alpha) = \{s' \in S | s \overset{\alpha}{\to} s'\}, \qquad Post(s) = \bigcup_{\alpha \in Act} Post(s, \alpha)$$

and the set of $\alpha - predecessors$ of s is expressed as:

$$Pre(s, \alpha) = \{s' \in S | s' \overset{\alpha}{\to} s\}, \qquad Pre(s) = \bigcup_{\alpha \in Act} Pre(s, \alpha)$$

We can extend this definition to a subset of S instead of a single state.

Definition 14 *Let TS be a transition system $C \subseteq S$ and $\alpha \in Act$, let:*

$$Post(C, \alpha) = \bigcup_{s \in C} Post(s, \alpha), \qquad Post(C) = \bigcup_{s \in C} Post(s)$$

and

$$Pre(C, \alpha) = \bigcup_{s \in C} Pre(s, \alpha), \qquad Pre(C) = \bigcup_{s \in C} Pre(s)$$

Example 28 *Let TS be the system in example 27. Some direct successors are $Post(Init, insert\_coin) = \{Select\}$ and $Post(Select) = \{Chips, Candy\}$. Some direct predecessors are $Pre(Candy, select\_candy) = \{Select\}$ and $Pre(Init) = \{Chips, Candy\}$.*

An *execution fragment* of a transition system is any alternating sequence of states and actions beginning and ending with a state. An *execution* ρ of a TS is an initial, maximal execution fragment. An initial fragment is a state/action sequence that starts in a initial state, *i.e.*, $s_0 \in I$ and a maximal execution fragment is whether a finite execution that ends in a terminal state, *i.e.*, s is a terminal state if and only if $Post(s) = \emptyset$, or an infinite state/action sequence.

Example 29 *An execution for the example 27 can be as follows, since it starts in a initial state and has infinite length:*

$$\begin{aligned}\rho = \quad & Init \xrightarrow{insert\_coin} Select \xrightarrow{select\_candy} Candy \xrightarrow{\tau} Init \xrightarrow{insert\_coin} Select \xrightarrow{select\_chips} \\ & \xrightarrow{select\_chips} Chips \xrightarrow{\tau} Init \xrightarrow{insert\_coin} Select \xrightarrow{select\_candy} Candy \xrightarrow{\tau} Init \cdots\end{aligned}$$

A finite (infinite) *path fragment* of a TS is a finite (infinite) state sequence $s_0 s_1 \ldots s_n$ ($s_0 s_1 s_2 \ldots$), such that $s_i \in Post(s_{i-1})$ for all $0 < i \leq n$ ($i > 0$). A *path* π of a TS is an initial, maximal path fragment. A path fragment is called initial if it stars in a initial state and is called maximal if it is either finite and ends in a terminal state or infinite.

Example 30 *A path fragment for the example 27 can be as follows, since it starts in a initial state and has infinite length:*

$$\pi = Init\, Select\, Candy\, Init\, Select\, Chips\, Init\, Select\, Candy\, Init \cdots$$

Definition 15 *A state s in a TS is reachable if there exists an initial, finite execution fragment that ends in s.*

Example 31 *For the TS described in example 27, all states are reachable.*

$$\begin{aligned} Select &= Init \xrightarrow{insert\_coin} Select \\ Candy &= Init \xrightarrow{insert\_coin} Select \xrightarrow{select\_candy} Candy \\ Chips &= Init \xrightarrow{insert\_coin} Select \xrightarrow{select\_chips} Chips \end{aligned}$$

A *trace* of a path fragment $Trace(\pi)$ is obtained when applied the labeling function to each state $s \in S$ forming the path fragment. For a path fragment $\pi = s_k s_{k+1} s_{k+2}$, $Trace(\pi) = L(s_k)L(s_{k+1})L(s_{k+2})$.

Example 32 *Let us consider the TS represented in Figure 3.12. Let π be a path fragment, as follows*

$$\pi = Select\, Candy\, Init\, Select\, Chips$$

the trace for this path fragment is

$$Trace(\pi) = \{paid\}\, \{selected\}\, \emptyset\, \{paid\}\, \{selected\}$$

Traces of a TS, $Traces(TS)$, are sequences of the form $L(s_0)L(s_1)L(s_2)\cdots$, where $s_i \in S$, $i \in \mathbb{Z}$ and there is a path π of the form $\pi = s_0s_1s_2\cdots$. They are words formed when applying the labeling function to a path belonging to the TS on the issue. For every path π in a TS, $Trace(\pi) \subseteq Traces(TS)$

Example 33 *Let TS be the transition system described in example 27. Let us analyze all possible paths. The system always starts at initial state Init and then goes to state Select. From this state, the system can go either to state Chips or Candy and then it returns to Init. Notice that their labeling functions result the same evaluation, $L(Chips) = L(Candy) = \{selected\}$. So, no matter what state the system reaches after Select, the trace for this TS is denoted by:*

$$Traces(TS) = \emptyset\,\{paid\}\,\{selected\}\,\emptyset\,\{paid\}\,\{selected\}\,\emptyset\,\{paid\}\,\{selected\}\,\emptyset\cdots$$

Definition 16 (Satisfaction Relation) *Let Φ be a propositional formula, then a state $s \in S$ satisfies Φ if $L(s)$ makes the formula true.*

$$s \models \Phi \quad iff \quad L(s) \models \Phi$$

In words, a state s satisfies a formula Φ if and only if the evaluation of $L(s)$ satisfies the formula Φ, i.e., makes it true, otherwise we say the state does not satisfy the formula, $s \not\models \Phi$.

Example 34 *Let us consider the TS described in example 27. Let us consider the formula $\Phi = \{selected\}$. The only states which satisfy the formula are Candy and Chips, for $L(Candy) = L(Chips) = \{selected\}$. So we say $Candy \models \Phi$ and $Chips \models \Phi$. The states Init and Select do not satisfy the formula Φ, so we say $Init \not\models \Phi$ and $Select \not\models \Phi$.*

Many properties of a TS can be expressed as temporal logic formulae, which will be explained in the following.

3.3.1 Linear temporal logic

One way to transcribe a property is by using the Linear Temporal Logic (LTL), which is a formalism to describe the behavior of the system. In temporal logic, time is not explicit as in hybrid systems, but the concept of time is related to the order of occurrence of a determinate formula or its components. LTL formulae are interpreted over paths, *i.e.*, linear sequences of states [27, 28]. It is an extension of propositional logic, therefore, we will introduce some basic concepts and syntax of the propositional logic before entering the LTL concepts.

Definition 17 (Propositional Logic) *Let AP be a set of atomic propositions. The set of propositional logic formulae over AP is inductively defined by the following rules:*

- *true is a formula;*
- *Any atomic proposition $a \in AP$ is a formula;*
- *If Φ_1, Φ_2 and Φ are formulae, then so are $(\neg\Phi)$ and $(\Phi_1 \wedge \Phi_2)$;*
- *Nothing else is a formula.*

The constant "*true*" stands for a proposition which holds in any context, independent of the interpretation of the atomic proposition. All other binary operators can be derived from the previous ones.

Let Φ_1, Φ_2 be formulae, then the following equivalences are true:

$\Phi_1 \vee \Phi_2 \equiv \neg(\neg\Phi_1 \wedge \neg\Phi_2)$ for *disjunction*

$\Phi_1 \rightarrow \Phi_2 \equiv \neg\Phi_1 \vee \Phi_2$ for *implication*

$\Phi_1 \leftrightarrow \Phi_2 \equiv (\neg\Phi_1 \wedge \neg\Phi_2) \vee (\Phi_1 \wedge \Phi_2)$ for *equivalence*

$\Phi_1 \otimes \Phi_2 \equiv (\neg\Phi_1 \wedge \Phi_2) \vee (\Phi_1 \wedge \neg\Phi_2)$ for *parity*

The syntax for propositional logic can be written in a more relaxed way as follows:

$$\Phi \quad ::= \quad true \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi$$

Saying a proposition holds means that an evaluation function, μ, assigns a truth value to each atomic proposition, *i.e.*, $\mu : AP \rightarrow \{0, 1\}$. We can represent each evaluation μ by a set $A_\mu = \{a \in AP | \mu(a) = 1\}$. We say $A_\mu \models \Phi$ if the evaluations of each atomic proposition given by A_μ makes the formula Φ true.

Example 35 *Let the set of atomic propositions $AP = \{a, b, c\}$, the evaluation function μ and $\mu(a) = \mu(b) = 1$ and $\mu(c) = 0$, which means, $A_\mu = \{a, b\}$. Let the formulae $\Phi_1 = (a \vee b) \wedge \neg c$ and $\Phi_2 = (\neg a \wedge c) \wedge b$. Then $A_\mu \models \Phi_1$ and $A_\mu \not\models \Phi_2$.*

Definition 18 (Linear Temporal Logic) *Let φ_1 and φ_2 be formulae and $a \in AP$, so a formula φ is defined as:*

- *true is a formula;*
- *Any atomic proposition $a \in AP$ is a formula;*

- *If φ_1, φ_2 and φ are formulae, then so are $(\neg\varphi)$, $(\varphi_1 \wedge \varphi_2)$, $(\bigcirc\varphi)$ and $(\varphi_1 \boldsymbol{U} \varphi_2)$;*

The syntaxes $\bigcirc\varphi$ and $\varphi_1\mathbf{U}\varphi_2$ stand for a formula that holds for the next step independently from the current interpretation of the atomic propositions and a formula φ_1 that holds until φ_2 holds true for the first time, respectively. The syntax for linear temporal logic can be written in a more relaxed way as follows:

$$\varphi \ ::= \ true \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1\mathbf{U}\varphi_2$$

Although time is not explicit in LTL, its concept is widely used in sentences that express that a sequence or state will *eventually*, *always* or *never* be reached. In this formalism are used atomic propositions, boolean connectives and special temporal operators to create expressions representing properties [27].

The $\Box$ operator is read as "Always" - something is satisfied now and forever in the future - and the $\Diamond$ operator is read as "Eventually" - something will eventually be satisfied.

As in propositional logic, we can obtain other connectives from the basic booleans previously described and we can obtain the temporal operators from the until operator, as follows:

$\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$ for *disjunction*

$\varphi_1 \rightarrow \varphi_2 \equiv \neg\varphi_1 \vee \varphi_2$ for *implication*

$\varphi_1 \leftrightarrow \varphi_2 \equiv (\neg\varphi_1 \wedge \neg\varphi_2) \vee (\varphi_1 \wedge \varphi_2)$ for *equivalence*

$\varphi_1 \otimes \varphi_2 \equiv (\neg\varphi_1 \wedge \varphi_2) \vee (\varphi_1 \wedge \neg\varphi_2)$ for *parity*

$\Diamond \equiv \text{true}\mathbf{U}\varphi$ for temporal operator *eventually*

$\Box\varphi \equiv \neg\Diamond\neg\varphi$ for temporal operator *always*

Example 36 *Let π be a path fragment, each circle representing a state and the set of atomic propositions $AP = \{a, b, c\}$. In the first line represented in Figure 3.13, for the formula $\varphi = a$ the labeling function for the current step must be a. For the second line, in order for the formula $\varphi = \bigcirc a$ to be true, the next step must satisfy a regardless of the evaluation of the current step. In the third line, the formula $\varphi = a\boldsymbol{U}b$ says that each step will evaluate a until it evaluates b. The fourth line, $\varphi = \Diamond a$, affirms that a must happen at least once during the path fragment. And the last line, $\varphi = \Box a$, assures that a must be true at every step. Note that when a state has more than one active atomic proposition implies that all of them are satisfied in that state, as in line 3, where both atomic propositions a and c are satisfied in that state.*

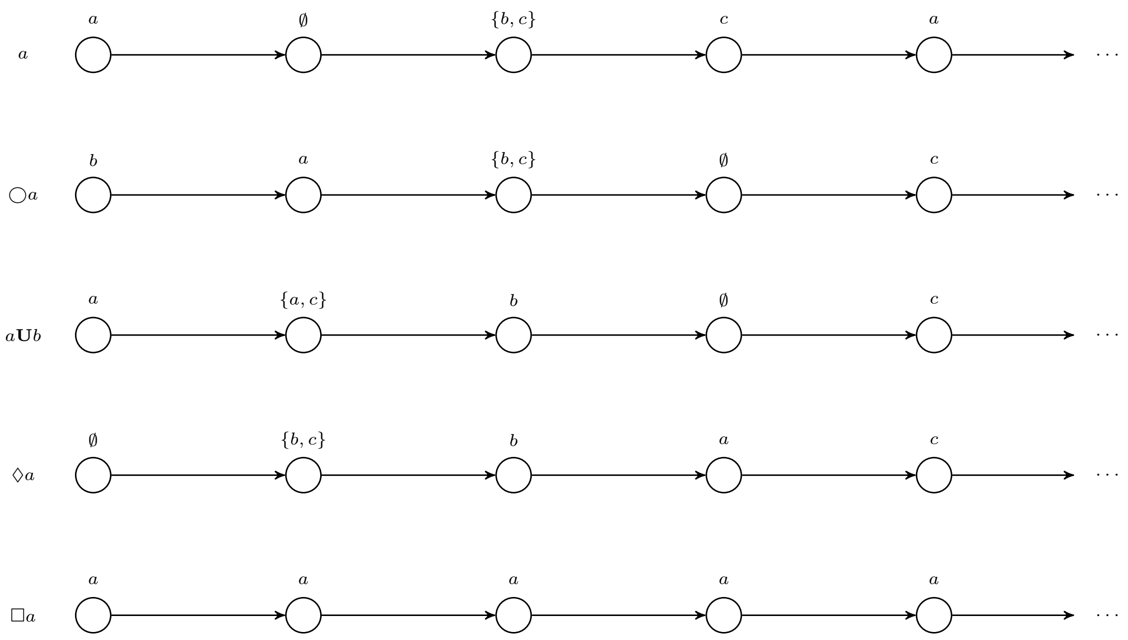


Figure 3.13: Example of LTL.

The LTL operators can be combined. For a formula φ, $\Box\Diamond\varphi$ is read as "infinitely often φ" and $\Diamond\Box$ is read as "eventually forever φ".

3.3.2 Linear-time properties

A system property can be understood as a specification of the system desired behavior. Some properties can be expressed as *linear-time properties*, which are a language of infinite words over the alphabet 2^{AP}.

Definition 19 (LT Property) *A linear-time property (LT property) over the set of atomic propositions AP is a subset of* $(2^{AP})^\omega$.

The set $(2^{AP})^\omega$ is the set obtained by the infinite concatenation of words in 2^{AP}. We will only consider systems without terminal states from this point. In order to a LT property be satisfied, it must meet the following relation.

Definition 20 (Satisfaction Relation for LT Properties) *Let P be an LT property over AP and* $TS = (S, Act, \rightarrow, I, AP, L)$ *a transition system without terminal states. TS satisfies P, TS* $\models$ *P, iff Traces(TS)* $\subseteq$ *P. A state s* $\in$ *S satisfies P, s* $\models$ *P, whenever Traces(s)* $\subseteq$ *P.*

Example 37 *Let us consider the TS described in example 27. Let us consider the formula* Φ *in the following*

Φ = *"You must pay the snack before you select it".*

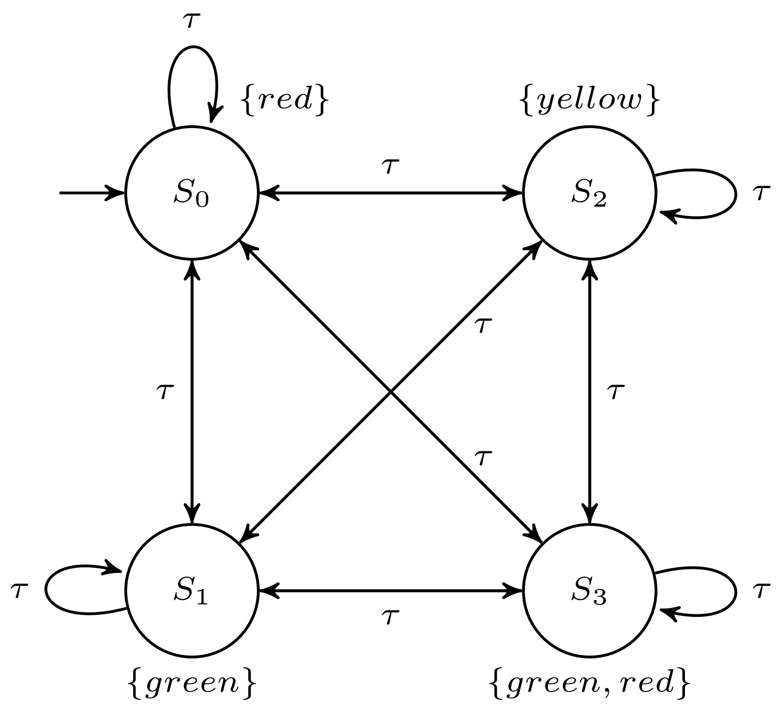


Figure 3.14: Transition system of Example 38.

Every time you have the atomic proposition $\{selected\}$ valid in a state it must have an atomic proposition $\{paid\}$ satisfied before it. For any path π of TS, $trace(\pi) = \emptyset \{paid\} \{selected\} \emptyset \{paid\} \{selected\} \emptyset \{paid\} \{selected\} \emptyset \cdots$ we verify the formula is satisfied. So $Trace(\pi)$ satisfies the formula Φ, therefore $TS \models P$.

Example 38 *Let us consider the transition system inspired in a traffic light depicted in Figure 3.14 where $AP = \{green, red\}$ is the set of atomic propositions and $S = \{S_0, S_1, S_2, S_3\}$ the set of states. Let the property P states:*

"The traffic light is infinitely often green".

The property P can be translated in the formula $\Phi = \Box\Diamond green$.

Then, among all possible traces, some of those which satisfy the property P are:

$$\{red\}\ \{yellow\}\ \{green, red\}\ \{red\}\ \{yellow\}\ \{green, red\}\ \{red\} \cdots$$
$$\{red\}\ \{green\}\ \{green\}\ \{green\}\ \{green\}\ \{green\} \cdots$$
$$\{red\}\ \{green\}\ \{yellow\}\ \{green\}\ \{yellow\}\ \{yellow\} \cdots$$

Notice, than in a finite computational time we may obtain a path $\pi_{fin} = S_0\ S_0\ S_0 \cdots\ S_0$, where $Trace(\pi_{fin}) = \{red\}\{red\}\{red\} \cdots \{red\}$. However, in an infinite computational time it is possible to obtain a new path π_{inf} for the continuation of π_{fin} where $Trace(\pi_{inf}) \models \Phi$, for example, $Trace(\pi_{inf}) = \{red\}\{red\}\{red\} \cdots \{red\}\{green\} \cdots \{green\} \cdots$. As we can observe, although a *finite trace gives the false impression of violating the property, there is nothing that prevents this trace continuation to fulfill the property. Therefore, the system satisfies the property [28].*

It is reasonable to expect that two transition systems with same traces satisfy the same LT properties. For this purpose, let us consider the following definition.

Theorem 2 (Trace Inclusion and LT Properties) *Let TS and TS' be transition systems without terminal states and with the same set of propositions AP. Then, for any LT property P, the following is true:*

$$Traces(TS) \subseteq Traces(TS') \text{ \textit{iff} } TS' \models P \text{ \textit{implies} } TS \models P$$

This proof can be found in [26].

Transition systems are said to be trace-equivalent if they have the same set of traces, as follows.

Definition 21 *Transition systems TS and TS' are trace-equivalent with respect to the set of propositions AP if* $Traces_{AP}(TS) = Traces_{AP}(TS')$.

Definition 21 directly implies in satisfaction relation.

$$Traces(TS) = Traces(TS') \text{ \textit{iff TS and TS' satisfy the same LT properties}}$$

The properties of a system characterize the type of the system. In the following we will distinguish some important LT properties.

Invariant property

An invariant property states that a condition Φ must hold for every reachable state.

Definition 22 (Invariant Properties) *An LT property* P_{inv} *over AP is an invariant if there is a propositional logic formula* Φ*, called an invariant condition of* P_{inv}*, over AP such that*

$$P_{inv} = \{A_0A_1A_2 \cdots \in (2^{AP})^{\omega} \mid \forall j \geq 0.\ A_j \models \Phi\}$$

Note that

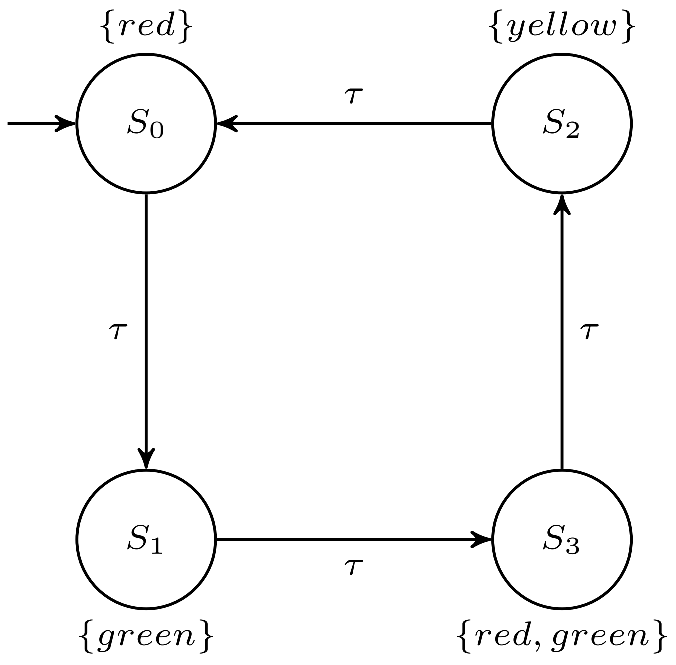


Figure 3.15: Transition system TS_{mod} of Example 39.

$$\begin{aligned} TS \models P_{inv} \quad & iff \quad trace(\pi) \in P_{inv}\ for\ all\ paths\ \pi\ in\ TS \\ & iff \quad L(s) \models \Phi\ for\ all\ states\ s\ that\ belong\ to\ a\ path\ of\ TS \\ & iff \quad L(s) \models \Phi\ for\ all\ states\ s \in Reach(TS). \end{aligned}$$

Example 39 *Let us take into consideration the transition system TS_{mod} depicted in figure 3.15. Consider the following invariant properties: Let us consider the traffic light example described in 39, .*

P_{inv_1} = *"The traffic light is always green"*

P_{inv_2} = *"The traffic light is never simultaneously green, red and yellow*

With the respective formulas:

$$\Phi_1 = \Box green$$
$$\Phi_2 = \neg(green \wedge red \wedge yellow)$$

The first property is not satisfied by TS_{mod}, since its initial state does not satisfy the property, $L(S_0) = \{red\}$, so the transition system does not satisfy the formula, $TS_{mod} \not\models \Phi_1$. The second property is satisfied by TS_{mod}, since every reachable state *holds the formula Φ_2, so $TS_{mod} \models \Phi_2$.*

Invariant properties are a particular kind of safety property and are also called state properties, therefore, they must be verified in every reachable state. In the following, we will explain the concept of safety property. Some safety properties require the verification to be based on a finite path fragment instead of each state individually.

Safety property

Safety properties can in an informal way be translated as "nothing bad should happen". We call *bad* states, the ones which violate the safety property. In other words, a safety property holds if a particular set of undesired or forbidden states is not reached by the system [7, 29].

Let us consider the traces $\sigma = \sigma_0\, \sigma_1\, \cdots\, \sigma_i \cdots$ and $\sigma' = \sigma'_0\, \sigma'_1\, \cdots\, \sigma'_i \cdots$ for the following definition.

Definition 23 (Safety Properties) *An LT property P_{safe} is a safety property if for every trace σ that violates P_{safe}, there exists an integer i, such that for every trace σ' that coincides with σ up to position i,i.e., $\forall 0 \leq j \leq i$, $\sigma'_i = \sigma_i$, σ' also violates ϕ.[28]*

In other words, in order to be a safety property, for all initial trace fragments that violate the property, called bad prefixes, their continuations also violate the property.

Let us consider $Traces_{fin}(TS)$ a set that contains all the finite traces of a transition system TS and $BadPref(P_{safe})$ a set that contains all the initial trace fragments which end in an element that violates the safety property. In that manner, the satisfaction relation for safety properties for a transition system TS without terminal states can be expressed as:

$$TS \models P_{safe} \text{ if and only if } Traces_{fin}(TS) \cap BadPref(P_{safe} = \emptyset)$$

An example of safety property is formula Φ represented in example 37.

Liveness

Liveness properties can in an informal way be translated as "something good will happen in the future". Unlike safety properties which can be violated in a computational finite time, liveness properties need an infinite trace as counterexample.

Definition 24 (Liveness Properties) *An LT property P_{live} is a liveness property if for every prefix of a trace $\sigma_0\, \sigma_1\, \cdots\, \sigma_i$ there exists an infinite trace σ, that starts with the same prefix $\sigma_0\, \sigma_1\, \cdots\, \sigma_i$ and $\sigma \models \phi$.[28]*

In other words even if the result obtained in a finite computation does not fulfill the property, it is still possible to find a continuation that will satisfy it. An example of safety property is formula Φ represented in example 38.

Chapter 4

Reachability-based diagnosability for hybrid systems

The diagnosability of a system is the ability to identify the occurrence of a failure by observing the behavior of the system after a finite time or a finite number of occurrences of events. The fault diagnosis of hybrid systems is an open field and we can identify that the usual approach is the combination of the purely discrete diagnosis of $L(H)$ with the distinction of the continuous dynamics for each location [1, 10]. BAYOUDH and TRAVÉ-MASSUYÈS [10] study seeks to obtain discrete information from the continuous dynamics using the DES diagnosis approach. The HS model is mapped into a DES infrastructure using mostly the DES tools for fault diagnosability verification. DIENE *et al.* [1] study uses DES techniques as well as continuous information for diagnosability verification. The work uses an adapted DES tool that is updated while the continuous information evolves.

In this work, we propose a new concept of diagnosability for hybrid systems combining the discrete diagnosis with the distinction of the continuous dynamics for each mode employing reachability analysis that is a suitable tool for Hybrid Systems. For this purpose some additional conditions are assumed on the elements of the hybrid automaton: the transition relation of the hybrid automaton is deterministic, the vector fields are globally continuous Lipschitz functions to guarantee the uniqueness of the solution of the differential equations that define the continuous dynamics in each location [12], and the invariant, the guard and the reset are linear functions on the components of the state.

This chapter is based on work [18], and is divided into two sections: in Section 4.1, we present a new HS diagnosability definition, and, in Section 4.2, we present a case study of fault diagnosability analysis of systems modeled by hybrid automata.

4.1 Diagnosability concept for hybrid systems

Let $\chi_0 \subseteq X$ be a set of initial conditions. In addition, we will consider the reachability within the same location, considering that it does not have possible transitions. We assume that all traces that do not contain the fault are called normal traces s_N and, equivalently, all traces that contain the fault are called faulty traces, s_F. All locations for which $q_0 \overset{s_N}{\rightarrow} q_N$ are called normal locations and locations $q_0 \overset{s_F}{\rightarrow} q_F$ are called fault locations. Let $\Psi(\Sigma_f)$ be the set of all sequences in L that end with a fault event σ_f, as defined in Definition 5 in Section 2.6. Let $\mathcal{R}_H(q_0, X_0)$ be the set of reachable states with respect to the continuous evolution, where $q_0 \in Q$ and the convex continuous set $X_0 \subseteq X$. Let n be the total number of steps and Ω_0 the overapproximation by template polyhedron of the initial set X_0.

In order to compute $\mathcal{R}_H$, we consider the affine continuous dynamics $\dot{x}(t) = Ax(t)+v(t)$, where $v(t) \in \mathbb{R}^n$ and can be expressed as $v(t) = Bu(t)$, where $B \in \mathbb{R}^{n\times m}$ and $u(t) \in \mathbb{R}^m$. Let $Reach_{t_k,t_{k+1}}(\Omega_{k-1})$ the reachable states starting from the set of continuous states Ω_{k-1} in the interval $[t_k, t_{k+1}]$ be denoted as:

$$Reach_{t_k,t_{k+1}}(\Omega_{k-1}) = \{x(\tau)|t_k \leq \tau \leq t_{k+1}, x(0) \in \Omega_{k-1}\}$$

where $x(t)$ satisfies the equation $\dot{x}(t) = Ax(t) + v(t)$. The set Ω_k is an overapproximation of the reachable set of continuous states $Reach_{t_k,t_{k+1}}(\Omega_{k-1})$. Thus, Ω_k covers the reachable states in the time interval $[t_k, t_{k+1}]$. The algorithm proposed in [16] calculates a sequence of continuous sets $\Omega_0, \cdots, \Omega_n$, called flowpipe, that is an overapproximation that covers all the calculated reachable states. In a simplified way,

$$\mathcal{R}_H(q_0, X_0) = \bigcup_{k=0,\cdots,n} \Omega_k$$

.

Definition 25 *[Diagnosability] Let L be the language generated by the hybrid automaton H. The hybrid system modeled by H is diagnosable with respect to projection P, set of faults Σ_f and the set of initial conditions $\chi_0 \subseteq X$ if conditions $D_D \vee D_C$ are satisfied, where*

$$\begin{aligned}
D_D :& (\exists n \in \mathbb{N})(\forall u \in \Psi(\Sigma_f))(\forall v \in L/u)(\|v\| \geq n) \\
& \Rightarrow (\nexists \omega \in L)[(P(uv) = P(\omega)) \wedge (\Sigma_f \notin \omega)] \\
D_C :& (\exists t_i \in \mathbb{R}^+)(\forall u \in \Psi(\Sigma_f)) \\
& (\exists v \in L/u \wedge v \in \Psi(\Sigma_o)) \\
& (P(uv) = P(\omega) \wedge \Sigma_f \notin \omega) \Rightarrow \\
& \mathcal{R}_H(\mathcal{Q}_\omega, \chi_\omega^{t_i}) \cap \mathcal{R}_H(\mathcal{Q}_{uv}, \chi_{uv}^{t_i}) = \emptyset
\end{aligned}$$

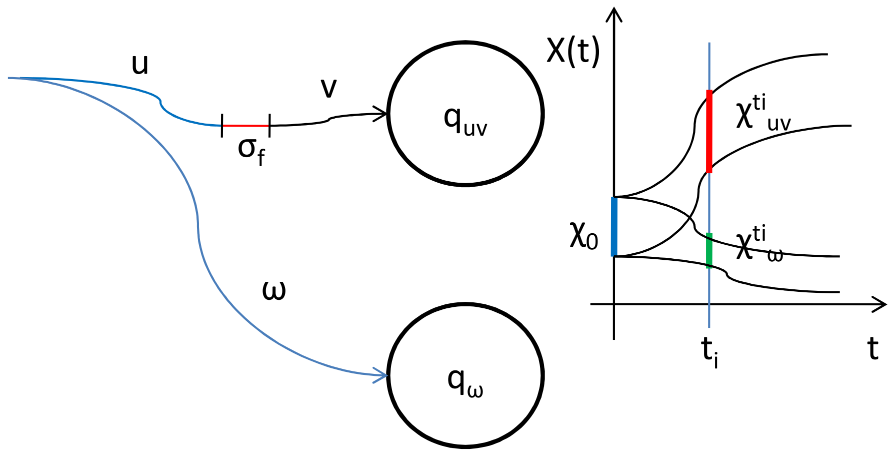


Figure 4.1: Illustration of the definition of hybrid systems diagnosability.

where $\mathcal{Q}_s = UR(q_0 \xrightarrow{s})$ *and* $\chi_s^{t_i} = \phi(\mathcal{Q}_s, \chi_0, t_i)$.

Definition 25 suggests that the system can be diagnosed either in a purely discrete event way or based on the continuous behavior of the hybrid system adopting the reachability approach. Condition D_D proposed in SAMPATH *et al.* [2] expresses that there must exist a trace of arbitrarily long length after the occurrence of the fault that is not confused with any normal trace of the system. Condition D_C expresses that when the faulty trace uv and a normal trace ω are ambiguous,*i.e.*, $P(uv) = P(\omega)$, there must exist at least one fault location that has its continuous behavior different from a normal location.

In DIENE *et al.* [1] work, Condition D_C for the continuous distinguishability of modes is accomplished with mode residual computation. The residual computation implements an observer of the continuous model, which is a continuous systems tool. In this work, we use an existent hybrid systems tool applied in the diagnosability context. This reachability tool is used in the continuous distinguishability of modes. Instead of comparing the residuals of the model, we analyze the evolution of the system based on the reachable regions sets. In order to be diagnosable, the system reachable regions sets of the fault and the normal modes must be disjoint at some finite time.

The reachability analysis is implemented using the tool called SpaceEx [16, 17]. This tool is a scalable reachability verification for hybrid systems. It uses the support function and template polyhedra representations of the continuous space state sets in order to compute an overapproximation of the reachable states.

Figure 4.1 illustrates the condition D_C, for the case when the fault location q_{uv} and normal location q_ω have different continuous evolutions with the same initial condition χ_0 from instant t_i. This difference is translated into the reachable sets

$\mathcal{R}_H(q_\omega, \chi_\omega^{t_i})$ and $\mathcal{R}_H(q_{uv}, \chi_{uv}^{t_i})$ with initial conditions $\chi_\omega^{t_i}$ and $\chi_{uv}^{t_i}$ having an empty intersection. Notice that the unobservable reach $\mathcal{Q}_s = UR(q_0 \xrightarrow{s})$ in condition D_C represents the set of all locations $\mathcal{Q}$, after the occurrence of s, which are reached by unobservable events, and $\chi_s^{t_i} = \phi(\mathcal{Q}_s, \chi_0, t_i)$ is a set of continuous states for solution $\dot{x} = f(q, x)$ in the interval (t_0, t_i) with initial condition χ_0 for all $q \in \mathcal{Q}_s$. It is worth mentioning that, implicit in condition D_C, is the need for the system to remain in the locations where this condition is being verified until condition $\mathcal{R}_H(\mathcal{Q}_\omega, \chi_\omega^{t_i}) \cap \mathcal{R}_H(\mathcal{Q}_{uv}, \chi_{uv}^{t_i}) = \emptyset$ is satisfied. If a new observable event occurs, conditions D_D and D_C must be checked again.

4.2 Example

In this section we will consider a hybrid system composed of a valve, a pump and a controller [19]. Consider a recipient with a leak where the valve flow is greater than the leak flow. The flow rate of the valve is subject to the pressure supplied by the pump. For simplicity, we represent the concurrent behavior of the hybrid automaton $H = (\Sigma, Q, E, Q_0, X, f, I, G, R, X_0)$ subject to the control action from the controller as highlighted in blue in Figure 4.2.

This HA represents the normal behavior of the system, where we can observe the set of events $\Sigma_o = \{ov, cv, SaP, SoP\}$, where ov consists of the command to open the valve, cv is the command to close the valve and SaP and SoP are the commands to start and stop the pump, respectively; the locations $Q = \{1, 2, 3, 4\}$; the continuous state f, representing the flow; the initial state $(Q_0, X_0) = (1, 0)$. The continuous dynamics at locations $1, 2$ and 4 is $\dot{f} = -9f$ and at location 3 is $\dot{f} = -9f + 25$. The invariant is $0 \leq f \leq 25/9$ for all locations. Location 1 consists of the pump stopped and the valve closed. When event ov occurs, it transitions to location 2, which consists of the valve opened and the pump stopped. In the occurrence of SaP, the automaton goes to location 3, where the valve is open and the pump is started. Analogously, when SoP occurs, the pump is stopped and the automaton is in location 4. The cycle closes after the valve closing command cv, returning to location 1.

Consider that this system is subject to the following failures associated with the events: fully stuck closed valve sc, fully stuck open valve so_{100} and valve 50% stuck from its total opening so_{50}, *i.e.*, set of fault events $\Sigma_f = \{sc, so_{100}, so_{50}\}$. By hypothesis, faults are unobservable events $\Sigma_f \subseteq \Sigma_{uo}$, which can occur at any time. So, the set of events is $\Sigma = \Sigma_o \cup \Sigma_f$. Figure 4.2 represents the full behavior of the system subject to the set of faults Σ_f, where the locations and transitions in blue represent the normal behavior of the system.

Let us analyze the fault trace where the fault event sc occurs for a possible trace

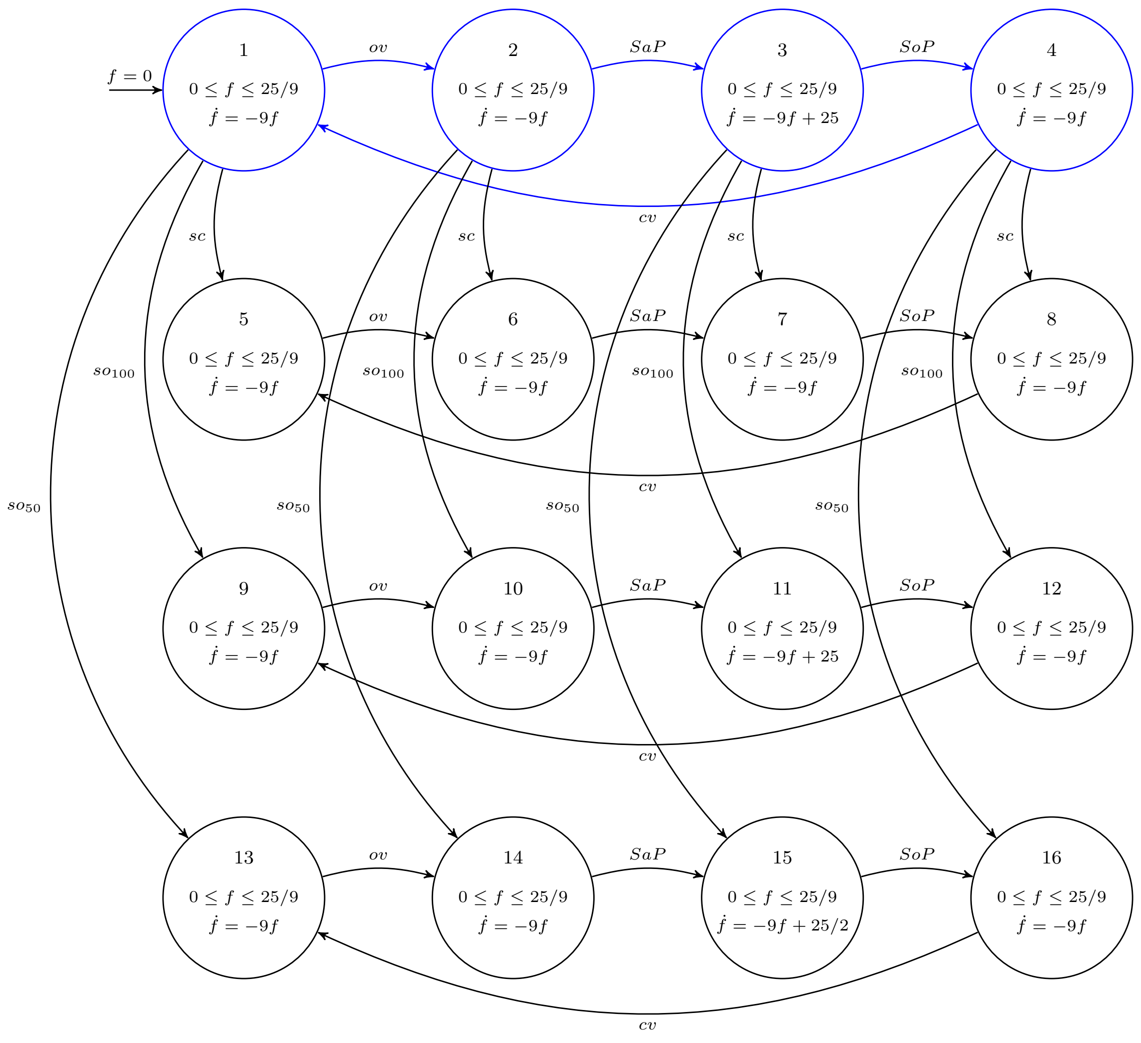
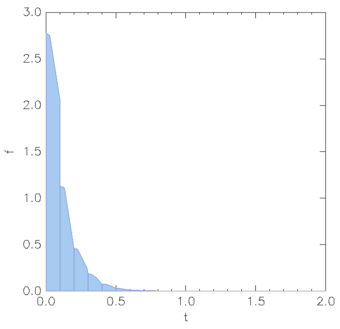
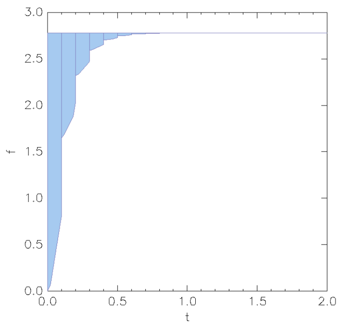


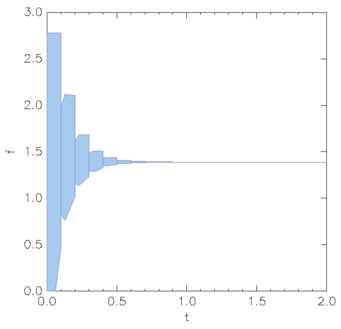
Figure 4.2: Example of the failure behavior represented by a hybrid automaton of Section 4.2.



(a)



(b)



(c)

Figure 4.3: Reachability chart related to locations of HA subject to the same initial conditions $\chi_0 = 0 \leq f \leq 25/9$ of the example in Section 4.2. Subfigure (a) refers to the behavior of locations $1, 2, 4-10, 12-14$ and 16. Subfigure (b) refers to the behavior of locations 3 and 11 and subfigure (c) refers to the behavior of location 15.

of the fault cycle $uv = \{sc\, ov\, SaP\, SoP\, cv\}^*$ and the trace of the normal cycle $\omega = \{ov\, SaP\, SoP\, cv\}^*$. In this case, we see that the language projection $P : \Sigma^* \to \Sigma_o^*$ of fault and normal traces are the same, *i.e.*, $P(uv) = \{ov\, SaP\, SoP\, cv\}^* = P(\omega)$. By inspection, we can verify that all fault cycles have the same projection as the normal cycle. Therefore, by Definition 25, the hybrid system modeled by H does not satisfy Condition D_D. To solve this problem, [19] proposes the use of sensor map. In this work, we will analyze diagnosability condition D_C that employs reachability analysis.

Figure 4.3 represents the reachability graphs for each location of the HA, considering that the reachability is performed subject to the same initial conditions $\chi_0 : 0 \leq f \leq 25/9$ at each location and it has no transitions. Time is limited by 2 seconds since all trajectories converge to a constant value within that period of time. Briefly, Figure 4.3-(a) represents the reachable regions for locations 1, 2, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14 and 16. Figure 4.3-(b) represents the reachable regions for locations 3 and 11 and Figure 4.3-(c) represents the reachable regions for location location 15.

Notice that from the first moment the fault event may have occurred, thus, by definition 25 a fault trace terminated in a fault sequence $u = sc$ and its possible continuation $v_1 = ov$ have the same projection as a normal trace ω_1, $P(uv_1) = ov = P(\omega_1)$. We must, then, find at least one reachable region of the fault locations that does not intersect the reachable region of normal locations from a time $t \geq t_i$. Locations related to these traces are $1 \xrightarrow{uv_1} 6$ and $1 \xrightarrow{\omega_1} 2$. The analysis is based on Figure 4.3-(a) taking into consideration the reachable regions of these locations, fault sc and normal behaviors, respectively. We can observe that $\nexists t_i \in \mathbb{R}^+$ that ensures the non-intersection of reachable regions.

In this manner, the continuation $v_2 = ov\,SaP$, whose projection $P(uv_2) = ov\,SaP$ is the same as the projection of the normal trace $\omega_2 = ov\,SaP$, $P(\omega_2) = ov\,SaP$. Locations related to these traces are $1 \xrightarrow{uv_2} 7$ and $1 \xrightarrow{\omega_2} 3$. By means of the graphs in Figures 4.3-(a) and (b) we verify that $\exists t_i \in \mathbb{R}^+$ where the reachable regions do not have intersection. By inspection, for example, $t_i = 0.2$ already satisfies condition D_C. We thus obtain the same result as the sensor map.

In the case of fault so_{100}, the sensor map is not able to differentiate between normal and fault behaviors. Also, by Definition 25, it is not possible to diagnose this fault since the reachable regions of normal and fault behaviors will always be the same as depicted in Figures 4.3-(a) and (b).

Differently from the case discussed in [19], a new fault was introduced, so_{50} which by the sensor map would not be distinguishable from the normal behavior of the system. Analyzing by Definition 25, let $u = so_{50}$ be a fault trace ended with fault event and its possible continuation $v_1 = ov$, projections $P(uv_1) = ov = P(\omega_1)$, where $\omega_1 = ov$. The locations related to these traces are $1 \xrightarrow{uv_1} 14$ and $1 \xrightarrow{\omega_1} 2$. We can observe that $\nexists t_i \in \mathbb{R}^+$ that ensures non-intersection of the reachable regions. For another continuation $v_2 = ov\,SaP$, whose projection $P(uv_2) = ov\,SaP$ is the same as the projection of the normal trace $\omega_2 = ov\,SaP$, $P(\omega_2) = ov\,SaP$ its referents locations are $1 \xrightarrow{uv_2} 15$ e $1 \xrightarrow{\omega_2} 3$ represented in Figures 4.3-(c) and (b), respectively. By inspection, $t_i = 0.2$ already satisfies the condition D_C. Thus, by employing the reachability analysis we can diagnose fault so_{50}.

4.3 Comparison example

As a comparison example, let us consider the underlying DES automaton G of Example 26 depicted in Figure 3.10. This system does not satisfy condition D_D, since for every faulty trace there is a normal trace with the same projection. We, thus, test condition D_C. Figures 4.4(a), (b), (c), (d) and (e) represent the reachability graphs of the models $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{M}_4$ and $\mathcal{M}_5$, respectively. The y-axis represents the angular position of the rudder, denoted as x_1, and the x-axis represents the time limited by 1 second. In the example, the angular position is in the interval $[0°, 35°]$. All the locations are subject to the same initial condition $\chi_0 = [0, 0.1, 0.1]^T$ with no transitions between locations.

Consider the automaton G of Example 26 depicted in Figure 3.10. Let us analyze a faulty trace that ends with a fault event σ_f, $u = ON\ ST_{ON}\ \sigma_f$. A possible continuation v_1 after the occurrence of the fault σ_f is $ST_{35^\circ}\ PO_{ON}$ which leads to location 9. The normal trace $\omega_1 = ON\ ST_{ON}\ ST_{35^\circ}\ PO_{ON}$, which leads to location 3, has the same projection as the faulty trace uv_1, $P(uv_1) = ON\ ST_{ON}\ ST_{35^\circ}\ PO_{ON} = P(\omega_1)$.

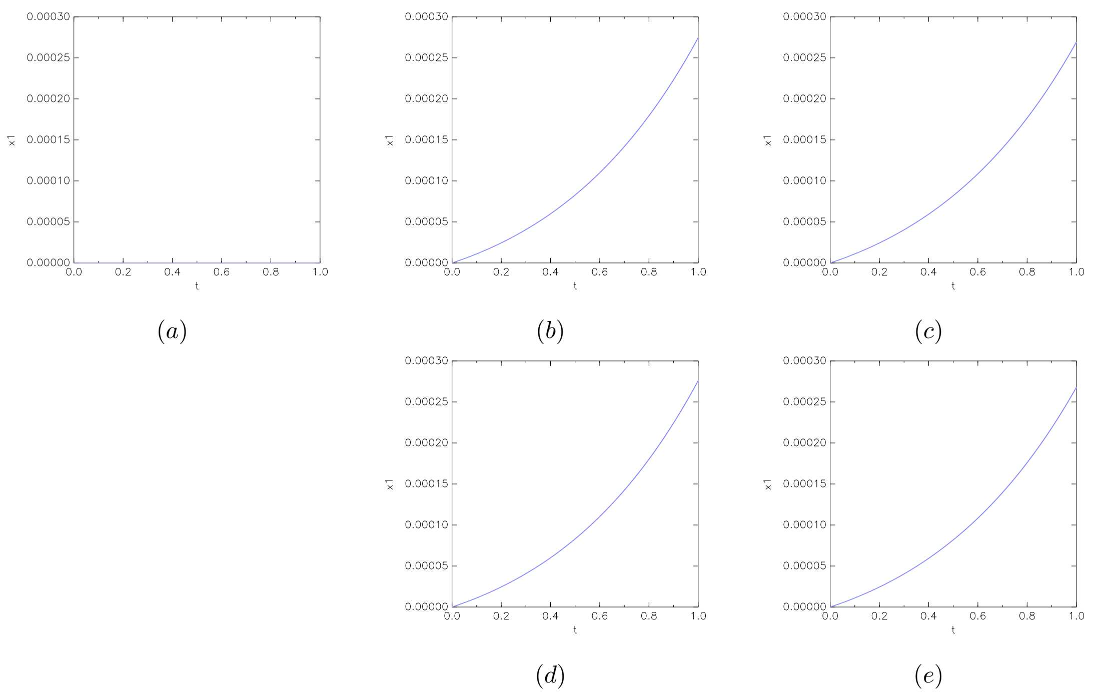


Figure 4.4: Reachability chart related to the models of the HA subject to the same initial condition $\chi_0 = [0, 0.1, 0.1]^T$ of the example in Section 4.3. Subfigures (a) - (e) refer to the behavior of the models $\mathcal{M}_1$, $\mathcal{M}_2$, $\mathcal{M}_3$, $\mathcal{M}_4$ and $\mathcal{M}_5$, respectively

Locations 3 and 9 are modeled by $\mathcal{M}_3$ and $\mathcal{M}_5$, respectively. Their reachable regions are presented in Figure 4.4(c) and (e), respectively. We use SpaceEx to calculate the reachable region of the locations, see Appendix A for more information. In order to be distinguishable, the reachable regions of this two locations must be different from a time t_i forward.

Not so clearly, we can observe that the regions are disjoint after 0.6 seconds, *i.e.*, according to Theorem 1, there exists a time t_i that satisfies Condition D_C. After $t_i = 0.6$ seconds, there is no intersection of the reachable regions of the locations. In that manner, modes 3 and 9 are distinguishable. The SpaceEx tool has a resource called "*forbidden states*", see Appendix A for more information. It is commonly used to check the system's safety. A system is safe if a given set of forbidden states is not reached, *i.e.*, not plotted. We use this resource to plot the regions where the reachable regions of the comparing models are the same.

For a more clear analysis, we use the forbidden states concept in Figure 4.5(a), that displays the reachable region of model $\mathcal{M}_3$ only when its region matches the reachable region of model $\mathcal{M}_5$. SpaceEx does not allow one to plot three or more variables in a same 2D graph, only two variables are allowed. In addition, it is not trivial to export SapaceEx data to be processed in another software. Thus, we use another approach to analyze the models behaviors depicted in Figure 4.5(b), where

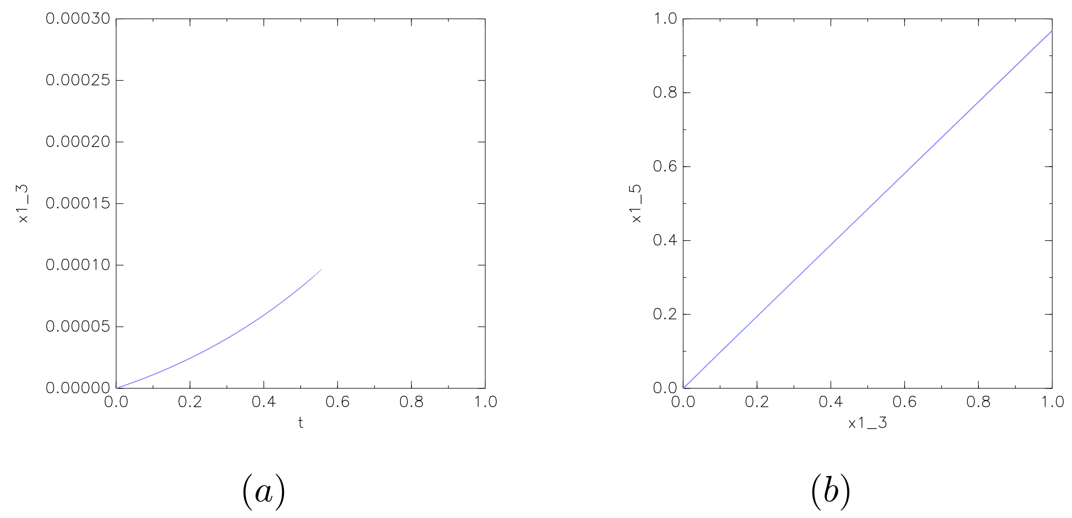


Figure 4.5: Reachability chart related to the models $\mathcal{M}_3$ and $\mathcal{M}_5$ of the example in Section 4.3. Figure (a) refers to the intersection of the reachable regions from the two models. This Figure displays the reachable region of model $\mathcal{M}_3$ when the region is the same as the reachable region of model $\mathcal{M}_5$. Figure (b) refers to the relation of the models $\mathcal{M}_3$ and $\mathcal{M}_5$ in the interval [0,1]

it displays the relation of the reachable regions between both models.

In a similar manner, the continuation $v_2 = RAP_{ON}\ RAP_{OFF}\ ST_{ON}$ leads to location 10. The normal trace $\omega_2 = ON\ ST_{ON}\ RAP_{ON}\ RAP_{OFF}\ ST_{ON}$, which leads to location 2, has the same projection as the faulty trace uv_2, $P(uv_2) = ON\ ST_{ON}\ RAP_{ON}\ RAP_{OFF}\ ST_{ON} = P(\omega_2)$. Locations 2 and 10 are modeled by $\mathcal{M}_2$ and $\mathcal{M}_4$, respectively. The reachable regions displayed in Figure 4.4 (b) and (d) are disjoint from 0.6 seconds forward. Thus, according to Theorem 1, exists a t_i that satisfies condition D_C. In that manner, locations 2 and 10 are distinguishable. For a more clear analysis, Figure 4.6(a) plots the reachable region of model $\mathcal{M}_2$ only when its region matches the reachable region of model $\mathcal{M}_4$. Figure 4.6(b) displays the relation of the reachable regions between both models.

Notice that for all other continuations, the traces reach one of those distinguishable pairs. In that manner, we can ensure diagnosability for this system according to Theorem 1, which is the same result found in [1].

Both methods follow normal and faulty discrete event paths that have same projection. They also try to distinguish the reached locations using continuous information. In order to accomplish this, they use different methods to distinguish their continuous behaviors. While in DIENE *et al.* [1] they use residual computation, in this work we use reachability analysis. One advantage of our approach is that it allows us to handle with a set of continuous initial conditions instead of a single one. In addition, in order to do the reachability analysis we use SpaceEx tool, that

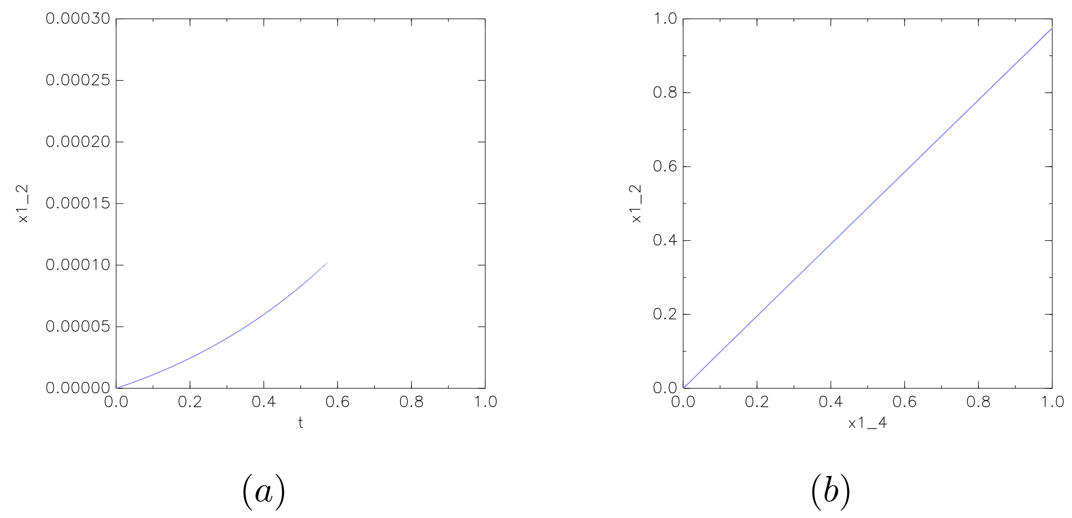


Figure 4.6: Reachability chart related to the models $\mathcal{M}_2$ and $\mathcal{M}_4$ of the example in Section 4.3. Figure (a) refers to the intersection of the reachable regions from the two models. This Figure displays the reachable region of model $\mathcal{M}_2$ when the region is the same as the reachable region of model $\mathcal{M}_4$. Figure (b) refers to the relation of the models $\mathcal{M}_2$ and $\mathcal{M}_4$ in the interval [0,1]

is a specific tool for hybrid systems, which may be a more natural approach to addressing this type of problem.

4.4 Final remarks

Throughout the study of the reachability analysis, we found some interesting issues, some positive and not so positive points. The SpaceEx and the reachability analysis concept give a different approach to the HS diagnosability, where it uses a scalable tool with few limitations regarding the number of modes. In addition, the SpaceEx tool is designed for hybrid systems. The idea is to use an existent HS tool applied to the HS diagnosability context.

In contrast, we found an issue regarding the region of the system stability. When both normal and fault modes reach the same landing, the definition proposed in this work consider them as undistinguishable modes. Even if the responses to a step input have different speeds, *i.e*, there may exist disjoint reachable regions in a limited time window, at infinite time they reach the same region.

A way to get around this issue is to implement a time window analysis. Our current analysis proposes that there must exist a time t_i, after which the reachable regions of the normal and fault modes are disjoint in order to be distinguishable. A time window analysis is a limited time band, an initial t_i and final time t_f. There must exist t_i and t_f such that, from the initial time until the final time, the reachable regions of the normal and fault modes are disjoint, despite reaching the same reachable region in the future. A recent work [30] proposes a similar approach for distinguishability analysis. The idea is the system input/output observation. The normal and faulty modes are distinguishable if at some time their reachable sets are disjoint.

Throughout the study of the reachability analysis we used the Simulink tool for HS simulation, as in Example 23. This tool provides a Stateflow chart object that represents a state machine with states, transitions, continuous dynamics, continuous transitions, among others. Within each state, we can describe the continuous dynamics associated with it. We can determine the continuous transitions, *i.e.*, guards, and initial conditions. In addition, it is possible to simulate two or more HA operating in parallel. However, it presents some issues. The tool does not take into account the invariant concept. In addition, discrete events do not work properly and, although the input may be randomly generated from a interval, it does not support a set of initial conditions. There does not yet exist a simulation tool complete and sufficient for Hybrid Systems diagnosability.

Chapter 5

Conclusion and future works

In this work, we present the main concepts related to hybrid systems, their properties and usual formal models. We propose a new definition of HS diagnosability combining DES diagnosability with reachability analysis to compare continuous behaviors. In addition, we present two definitions and methods in this research field that allow us to compare the existing works related to HS diagnosability.

Furthermore, we present a case study of fault diagnosability analysis of systems modeled by hybrid automata. This example demonstrates the advantage of performing the state reachability analysis associated with the continuous-time dynamics of the hybrid model. With this approach, it is possible to diagnose failures that would not be possible using only DES techniques.

In order to present the HS definition, we need some preliminary concepts of Chapter 2, as the concept of DES diagnosability presented in Section 2.6. In addition, it is necessary the formalism of the hybrid automata in Section 3.1 to introduce some existing methods described in Section 3.3. We formalize our definition in Section 4.1 along with the case study of fault diagnosability analysis in Section 4.2, a comparison example with an existent HS diagnosability method in Section 4.3 and some final remarks in Section 4.4.

Notice that HS diagnosability is not a consolidated research field and there is not a unique method. Reachability analysis may not be the ultimate method, however, it is a different approach to this growing field.

Future Works

As a future work perspective, we suggest to deepen the use of reachability analysis of hybrid systems in order to achieve diagnosability verification. In addition, the construction of an online diagnoser using a prior analysis of the system reachability to determine the normal regions.

Moreover, another way to improve the reachability analysis would be to propose a less restrictive HS diagnosability definition taking into consideration the time window approach.

Bibliography

[1] DIENE, O., MOREIRA, M. V., SILVA, E. A., et al. “Diagnosability of hybrid systems”, *IEEE Transactions on Control Systems Technology*, , n. 99, pp. 1–8, 2017.

[2] SAMPATH, M., SENGUPTA, R., LAFORTUNE, S., et al. “Diagnosability of discrete-event systems”, *IEEE Transactions on automatic control*, v. 40, n. 9, pp. 1555–1575, 1995.

[3] ZAYTOON, J., LAFORTUNE, S. “Overview of fault diagnosis methods for discrete event systems”, *Annual Reviews in Control*, v. 37, n. 2, pp. 308–320, 2013.

[4] HENZINGER, T. A. “The theory of hybrid automata”. In: *Verification of Digital and Hybrid Systems*, n. 170, NATO ASI Series (Series F: Computer and Systems Sciences), Springer-Berlin-Heidelberg, pp. 265–292, 2000.

[5] RASKIN, J.-F. “An introduction to hybrid automata”. In: Springer (Ed.), *Handbook of Networked and Embedded Control Systems*, pp. 491–517, 2005.

[6] GOEBEL, R., HESPANHA, J., TEEL, A. R., et al. “Hybrid systems: generalized solutions and robust stability”, *IFAC Proceedings Volumes*, v. 37, n. 13, pp. 1–12, 2004.

[7] FREHSE, G., KROGH, B. H., RUTENBAR, R. A. “Verifying analog oscillator circuits using forward/backward abstraction refinement”. In: *Proceedings of the conference on Design, automation and test in Europe: Proceedings*, pp. 257–262. European Design and Automation Association, 2006.

[8] HARIRCHI, F., OZAY, N. “Model invalidation for switched affine systems with applications to fault and anomaly detection”, *IFAC-PapersOnLine*, v. 48, n. 27, pp. 260–266, 2015.

[9] HARIRCHI, F., LUO, Z., OZAY, N. “Model (in) validation and fault detection for systems with polynomial state-space models”. In: *American Control Conference (ACC), 2016*, pp. 1017–1023. IEEE, 2016.

[10] BAYOUDH, M., TRAVÉ-MASSUYÈS, L. "Diagnosability analysis of hybrid systems cast in a discrete-event framework", *Discrete Event Dynamic Systems*, v. 24, n. 3, pp. 309–338, 2014.

[11] DIENE, O., MOREIRA, M. V., ALVAREZ, V. R., et al. "Computational methods for diagnosability verification of hybrid systems". In: *Control Applications (CCA), 2015 IEEE Conference on*, pp. 382–387. IEEE, 2015.

[12] TRIPAKIS, S., DANG, T. "Modeling, verification and testing using timed and hybrid automata", *Model-Based Design for Embedded Systems*, pp. 383–436, 2009.

[13] ALUR, R., COURCOUBETIS, C., HALBWACHS, N., et al. "The algorithmic analysis of hybrid systems", *Theoretical computer science*, v. 138, n. 1, pp. 3–34, 1995.

[14] MALER, O. "Algorithmic verification of continuous and hybrid systems". In: *Int. Workshop on Verification of Infnite-State System (Infnity)*, 2013.

[15] LE GUERNIC, C. *Reachability analysis of hybrid systems with linear continuous dynamics.* Computer science, Université Joseph-Fourier - Grenoble I, 2009.

[16] GORAN, F., LE GUERNIC, C., DONZÉ, A., et al. "SpaceEx: Scalable Verification of Hybrid Systems". In: *Computer Aided Verification. CAV*, v. 6806, *Lecture Notes in Computer Science*, pp. 379–395. Springer, Berlin, Heidelberg, 2011.

[17] FREHSE, G. "An introduction to spaceex v0. 8", *December*, 2010.

[18] S. VIEIRA, J., K. CARVALHO, L., V. L. NUNES, E., et al. "Diagnosticabilidade de sistemas híbridos empregando anÃ¡lise de alcançabilidade". In: *Proceedings of the XXII Congresso Brasileiro de Automática, João Pessoa*, p. xx, 2018.

[19] CASSANDRAS, C. G., LAFORTUNE, S. *Introduction to discrete event systems.* Springer Science & Business Media, 2009.

[20] BASILIO, J. C., CARVALHO, L. K., MOREIRA, M. V. "Diagnose de falhas em sistemas a eventos discretos modelados por autômatos finitos", *Revista Controle & Automaçao*, v. 21, n. 5, pp. 510–533, 2010.

[21] VIANA, G. S., BASILIO, J. C. "Codiagnosability of discrete event systems revisited: A new necessary and sufficient condition and its applications", *Automatica*, v. 101, pp. 354–364, 2019.

[22] LIBERZON, D. *Switching in systems and control.* Springer Science & Business Media, 2003.

[23] CHUTINAN, A. "Hybrid system verification using discrete model approximations", *Ph. D. dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University*, 1999.

[24] VAN DER SCHAFT, A. J., SCHUMACHER, J. M. *An introduction to hybrid dynamical systems*, v. 251. Springer London, 2000.

[25] BOYD, S., VANDENBERGHE, L. *Convex optimization.* Cambridge university press, 2004.

[26] BAIER, C., KATOEN, J.-P. *Principles of Model Checking.* MIT Press, 2008.

[27] CLARKE, E. M., GRUMBERG, O., PELED, D. *Model checking.* MIT press, 1999.

[28] CLARKE, E. M., HENZINGER, T. A., VEITH, H., et al. *Handbook of model checking.* Springer, 2018.

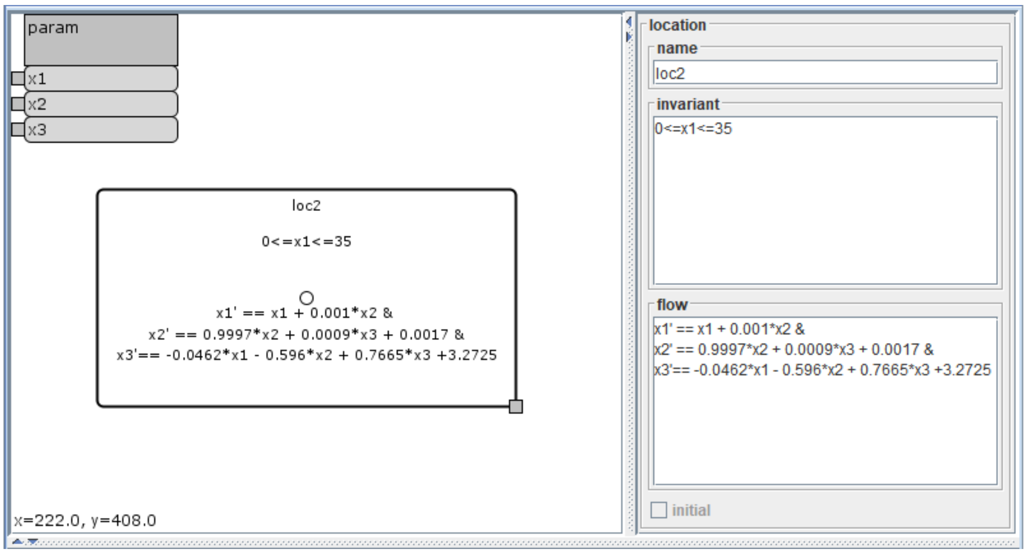
[29] HENZINGER, T. A., HO, P.-H., WONG-TOI, H. "HyTech: A model checker for hybrid systems", *International Journal on Software Tools for Technology Transfer*, v. 1, n. 1-2, pp. 110–122, 1997.

[30] YANG, L., OZAY, N. "Combining LTL monitoring with model invalidation for improved fault detectability analysis for hybrid systems". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pp. 278–279. ACM, 2019.

Appendix A

SpaceEx

In this chapter we illustrate the use of the Software SpaceEx[16, 17], as some codes used in this work and the tool's structure. The models are created with the model editor and the graphs are generated with the web interface. As an example, let us consider the example described in Section 4.3.



```
loc2
0<=x1<=35
x1' == x1 + 0.001*x2 &
x2' == 0.9997*x2 + 0.0009*x3 + 0.0017 &
x3'== -0.0462*x1 - 0.596*x2 + 0.7665*x3 +3.2725
```

name: loc2

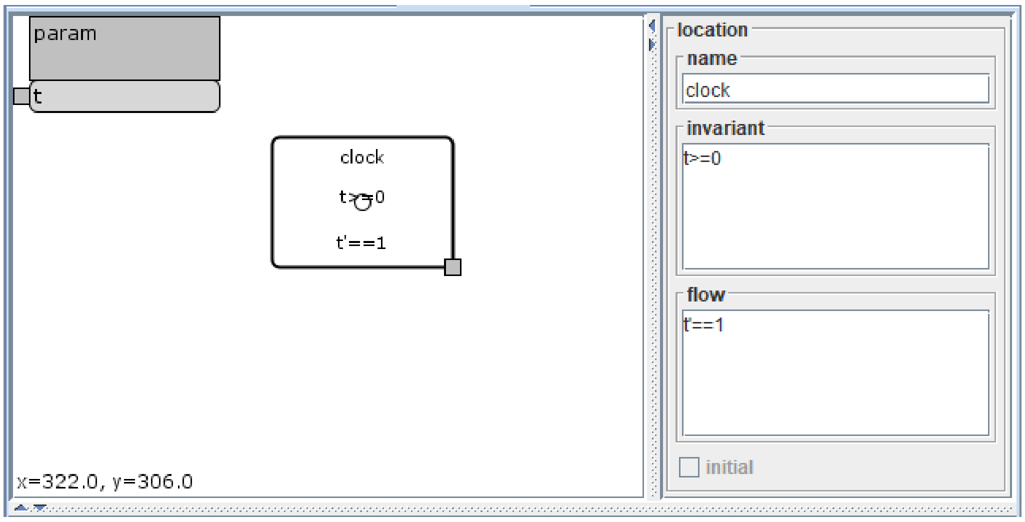
invariant:
```
0<=x1<=35
```

flow:
```
x1' == x1 + 0.001*x2 &
x2' == 0.9997*x2 + 0.0009*x3 + 0.0017 &
x3'== -0.0462*x1 - 0.596*x2 + 0.7665*x3 +3.2725
```

☐ initial

Figure A.1: Dynamics of model $\mathcal{M}_2$ of the example in Section 4.3.

In the model editor interface we declare the state variables and constants in an element called *base component*, which corresponds to a hybrid automaton. Each location has its own *name*, *invariant* and *flow*. For the example, we constructed five files containing the five models, a model per file. Let us consider the model $\mathcal{M}_2$ represented in Figure A.1. The other models have analogous construction. For the simulation, we consider the reachable states inside one location, without transitions. Model $\mathcal{M}_2$ location's name is “loc2” and only the variables x_1, x_2 and x_3 need to be declared. All locations have the same invariant, which is $0 \leq x_1 \leq 35$, the angular position constrain and each location has its own particular flow, given by



name: clock

invariant: t>=0

flow: t'==1

Figure A.2: Dynamics of the variable representing the time for the example in Section 4.3.

the HA vector field. The model does not support the matrix/vector notation and each variable derivative expression, vector field, is separated by the & character. The derivative is represented by the prime behind the variable.

Each file contains another base component named clock that models time, see Figure A.2. It is necessary two variables for a 2D graph in SpaceEx, thus the time variable must be declared, in order to be plotted. This variable could be inserted in the same location as the model location, but for organizational purpose, we constructed two different base components that can run in parallel, when declared in a *network component*. A network component is where is specified which base components run in parallel, *i.e.*, the parallel composition of the hybrid automata.

Figure A.3 displays the SpaceEx web interface. In the model tab we can select the model created in the model editor and a configuration file, whether it exists. The configuration file contains some parameters for the simulation and the initial state values, we generate a file by saving the current parameter configuration. Let us consider the simulation of the model $\mathcal{M}_2$ of the example described in Section 4.3.

Figure A.4(a) displays the model tab, where we select the file corresponding to the model $\mathcal{M}_2$ and a configuration file. Figure A.4(b) displays the specification tab, where we select which base or network component will be simulated. The initial states field is the configuration of the initial values of the system, in this case, $x_0 = [0, 0.1, 0.1]^T$ and initial time $t_0 = 0$. The forbidden states field is an expression that dictates an unsafe region. The 2D graph will plot the intersection between the forbidden states and the reachable region of the output variables, see Figure A.5(b). Figure A.5(a) displays the options tab, where we can select the type of scenario, *i.e.*, the verification algorithm and its set representation. We use de LGG Support Function that supports piecewise affine dynamics. The directions correspond to the

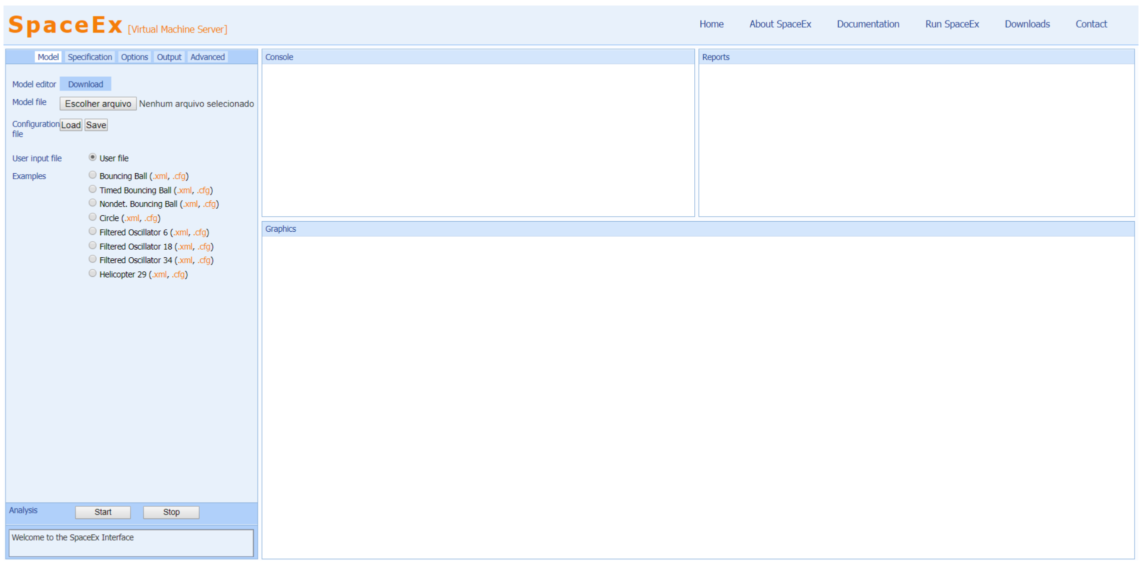
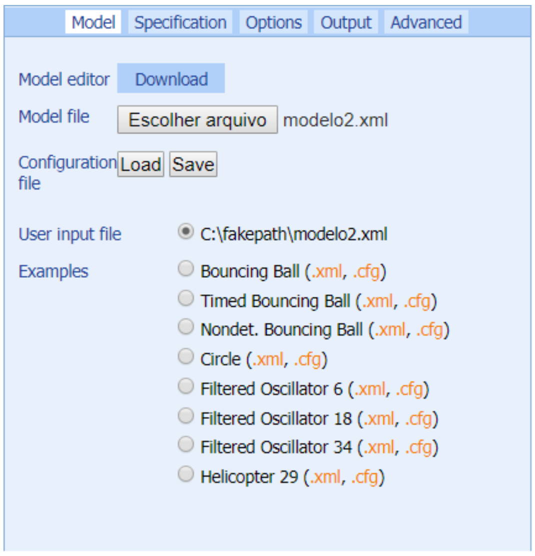

SpaceEx [Virtual Machine Server]
Home
About SpaceEx
Documentation
Run SpaceEx
Downloads
Contact
Model
Specification
Options
Output
Advanced
Model editor
Download
Model file
Escolher arquivo
Nenhum arquivo selecionado
Configuration file
Load
Save
User input file
User file
Examples
Bouncing Ball (.xml, .cfg)
Timed Bouncing Ball (.xml, .cfg)
Nondet. Bouncing Ball (.xml, .cfg)
Circle (.xml, .cfg)
Filtered Oscillator 6 (.xml, .cfg)
Filtered Oscillator 18 (.xml, .cfg)
Filtered Oscillator 34 (.xml, .cfg)
Helicopter 29 (.xml, .cfg)
Analysis
Start
Stop
Welcome to the SpaceEx Interface
Console
Reports
Graphics

Figure A.3: SpaceEx web interface homepage.

overapproximation set representation, we use the uniform representations, which are uniform directions. Clustering percentage and aggregating sets correspond to the percentage and the form of aggregation. The sampling time chosen as $T_S = 0.001$. The local time horizon is the simulation time, we use 1 second. And last, the maximum number of iterations, which represents the maximum number of discrete transitions. There are no transitions in the reachability analysis, a negative value gives no limitation for this parameter.

Figure A.5(b) displays the output tab, where we can select the output format: 2D graph, 3D graph, interval bounds and text. The 2D graph is the one used for plotting the reachable region, it is a two variable plot. The 3D graph is a three variable plot. Interval bounds gives the limits of the simulation for each variable, the minimum and maximum values reached. The text format gives the b coefficient and the template directions of the template polyhedra that overapproximates each reachable set. As defined in Section 3.1.2, a template polyhedron is given by $P_D = \{x \in \mathbb{R}^n | \bigwedge_{\ell_i \in D} \ell_i \cdot x \leq b_i\}$, where $D = \{\ell_1, \cdots, \ell_m\}$ is the set of template directions. For generating the text in Figure A.6, we substitute the local time horizon for 0.005, for a better understanding. The number of lines is the same as the number of template directions, in this case, 36. Each line contains the set of coefficients for the respective direction vector. The number of coefficients at each line is the number of steps ("local time horizon" \ "sampling time") of the simulation. For example, for the first step, we use the first column of coefficients and the first set of inequalities would be:

$$-x_1 - 0.287033x_2 + 0.738102x_3 - 0.941383t \leq 0.0465651$$



Model editor: Download

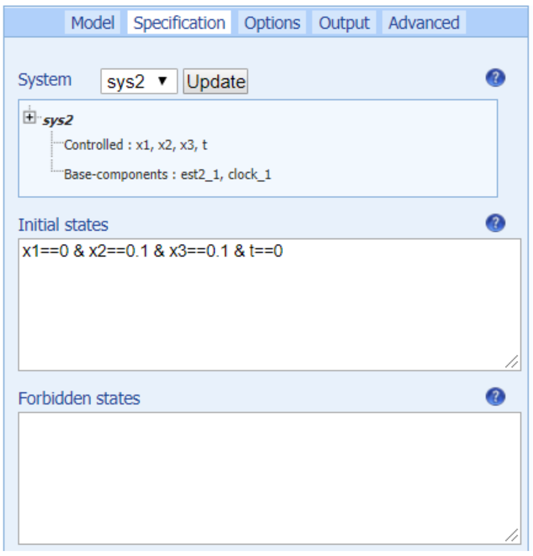
Model file: Escolher arquivo modelo2.xml

Configuration file: Load Save

User input file: [✓ C:\fakepath\modelo2.xml]

Examples:
- Bouncing Ball (.xml, .cfg)
- Timed Bouncing Ball (.xml, .cfg)
- Nondet. Bouncing Ball (.xml, .cfg)
- Circle (.xml, .cfg)
- Filtered Oscillator 6 (.xml, .cfg)
- Filtered Oscillator 18 (.xml, .cfg)
- Filtered Oscillator 34 (.xml, .cfg)
- Helicopter 29 (.xml, .cfg)

(a)



System: sys2 Update

```
sys2
  Controlled : x1, x2, x3, t
  Base-components : est2_1, clock_1
```

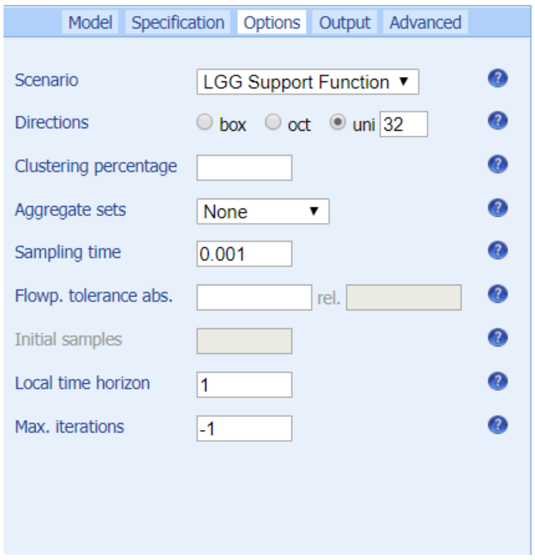
Initial states

```
x1==0 & x2==0.1 & x3==0.1 & t==0
```

Forbidden states

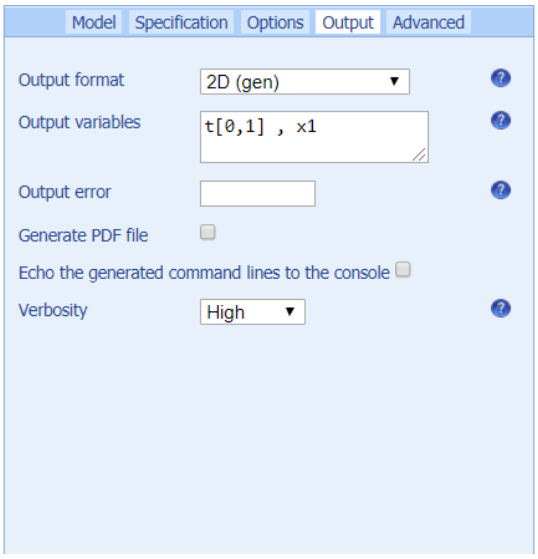
(b)

Figure A.4: SpaceEx model tab, (a). SpaceEx specification tab, (b).



Scenario	LGG Support Function
Directions	box oct [✓ uni] 32
Clustering percentage	
Aggregate sets	None
Sampling time	0.001
Flowp. tolerance abs.	rel.
Initial samples	
Local time horizon	1
Max. iterations	-1

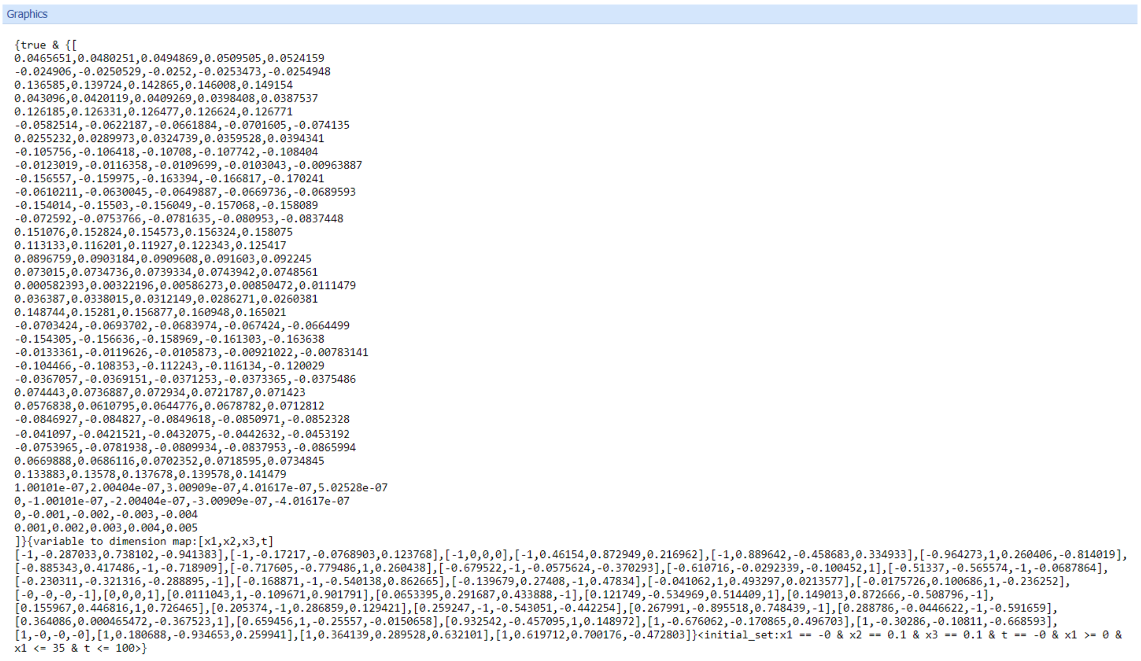
(a)



Output format	2D (gen)
Output variables	`t[0,1] , x1`
Output error	
Generate PDF file	☐
Echo the generated command lines to the console	☐
Verbosity	High

(b)

Figure A.5: SpaceEx options tab, (a). SpaceEx output tab, (b).



Graphics

```
{true & {[
0.0465651,0.0480251,0.0494869,0.0509505,0.0524159
-0.024906,-0.0250529,-0.0252,-0.0253473,-0.0254948
0.136585,0.139724,0.142865,0.146008,0.149154
0.043096,0.0420119,0.0409269,0.0398408,0.0387537
0.126185,0.126331,0.126477,0.126624,0.126771
-0.0582514,-0.0622187,-0.0661884,-0.0701605,-0.074135
0.0255232,0.0289973,0.0324739,0.0359528,0.0394341
-0.105756,-0.106418,-0.10708,-0.107742,-0.108404
-0.0123019,-0.0116358,-0.0109699,-0.0103043,-0.00963887
-0.156557,-0.159975,-0.163394,-0.166817,-0.170241
-0.0610211,-0.0630045,-0.0649887,-0.0669736,-0.0689593
-0.154014,-0.15503,-0.156049,-0.157068,-0.158089
-0.072592,-0.0753766,-0.0781635,-0.080953,-0.0837448
0.151076,0.152824,0.154573,0.156324,0.158075
0.113133,0.116201,0.11927,0.122343,0.125417
0.0896759,0.0903184,0.0909608,0.091603,0.092245
0.073015,0.0734736,0.0739334,0.0743942,0.0748561
0.000582393,0.00322196,0.00586273,0.00850472,0.0111479
0.036387,0.0338015,0.0312149,0.0286271,0.0260381
0.148744,0.15281,0.156877,0.160948,0.165021
-0.0703424,-0.0693702,-0.0683974,-0.067424,-0.0664499
-0.154305,-0.156636,-0.158969,-0.161303,-0.163638
-0.0133361,-0.0119626,-0.0105873,-0.00921022,-0.00783141
-0.104466,-0.108353,-0.112243,-0.116134,-0.120029
-0.0367057,-0.0369151,-0.0371253,-0.0373365,-0.0375486
0.074443,0.0736887,0.072934,0.0721787,0.071423
0.0576838,0.0610795,0.0644776,0.0678782,0.0712812
-0.0846927,-0.084827,-0.0849618,-0.0850971,-0.0852328
-0.041097,-0.0421521,-0.0432075,-0.0442632,-0.0453192
-0.0753965,-0.0781938,-0.0809934,-0.0837953,-0.0865994
0.0669888,0.0686116,0.0702352,0.0718595,0.0734845
0.133883,0.13578,0.137678,0.139578,0.141479
1.00101e-07,2.00404e-07,3.00909e-07,4.01617e-07,5.02528e-07
0,-1.00101e-07,-2.00404e-07,-3.00909e-07,-4.01617e-07
0,-0.001,-0.002,-0.003,-0.004
0.001,0.002,0.003,0.004,0.005
]}{variable to dimension map:[x1,x2,x3,t]
[-1,-0.287033,0.738102,-0.941383],[-1,-0.17217,-0.0768903,0.123768],[-1,0,0,0],[-1,0.46154,0.872949,0.216962],[-1,0.889642,-0.458683,0.334933],[-0.964273,1,0.260406,-0.814019],
[-0.885343,0.417486,-1,-0.718909],[-0.717605,-0.779486,1,0.260438],[-0.679522,-1,-0.0575624,-0.370293],[-0.610716,-0.0292339,-0.100452,1],[-0.51337,-0.565574,-1,-0.0687864],
[-0.230311,-0.321316,-0.288895,-1],[-0.168871,-1,-0.540138,0.862665],[-0.139679,0.27408,-1,0.47834],[-0.041062,1,0.493297,0.0213577],[-0.0175726,0.100686,1,-0.236252],
[-0,-0,-0,-1],[0,0,0,1],[0.0111043,1,-0.109671,0.901791],[0.0653395,0.291687,0.433888,-1],[0.121749,-0.534969,0.514409,1],[0.149013,0.872666,-0.508796,-1],
[0.155967,0.446816,1,0.726465],[0.205374,-1,0.286859,0.129421],[0.259247,-1,-0.543051,-0.442254],[0.267991,-0.895518,0.748439,-1],[0.288786,-0.0446622,-1,-0.591659],
[0.364086,0.000465472,-0.367523,1],[0.659456,1,-0.25557,-0.0150658],[0.932542,-0.457095,1,0.148972],[1,-0.676062,-0.170865,0.496703],[1,-0.30286,-0.10811,-0.668593],
[1,-0,-0,-0],[1,0.180688,-0.934653,0.259941],[1,0.364139,0.289528,0.632101],[1,0.619712,0.700176,-0.472803]}<initial_set:x1 == -0 & x2 == 0.1 & x3 == 0.1 & t == -0 & x1 >= 0 &
x1 <= 35 & t <= 100>}
```

Figure A.6: SpaceEx text format output.

first template direction and the first coefficient of the first line, until the last template direction and the first coefficient of the last line:

$$x_1 + 0.619712x_2 + 0.700176x_3 - 0.472803t \leq 0.001$$

For the second step, we use the second column of coefficients and the second set of inequalities would be:

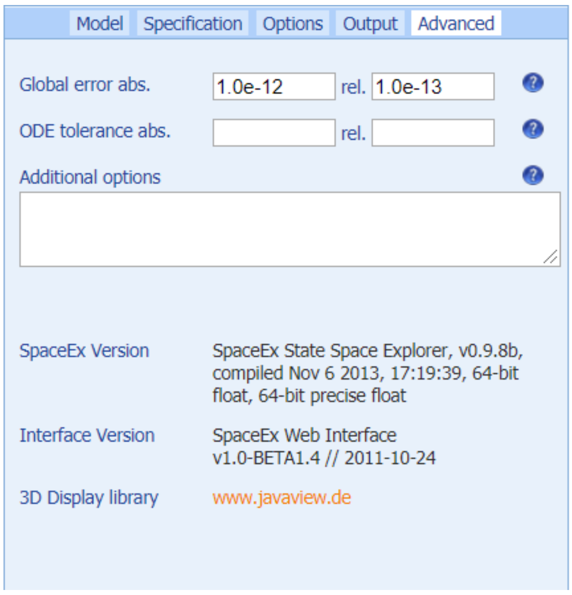
$$-x_1 - 0.287033x_2 + 0.738102x_3 - 0.941383t \leq 0.0480251$$

first template direction and the second coefficient of the first line, until the last template direction and the second coefficient of the last line.

$$x_1 + 0.619712x_2 + 0.700176x_3 - 0.472803t \leq 0.002$$

.

Figure A.7 displays the advanced tab, where it is configured the absolute and relative error for floating point computations and absolute and relative error for ODE solver.



Global error abs.	1.0e-12 rel. 1.0e-13
ODE tolerance abs.	rel.
Additional options	

SpaceEx Version	SpaceEx State Space Explorer, v0.9.8b, compiled Nov 6 2013, 17:19:39, 64-bit float, 64-bit precise float
Interface Version	SpaceEx Web Interface v1.0-BETA1.4 // 2011-10-24
3D Display library	www.javaview.de

Figure A.7: SpaceEx advanced tab.

Appendix B

Simulink - Chart

In this chapter we illustrate the use of Chart object in Software Simulink. The models are created within the chart object and the graphs can be analyzed in the scope object or being generated with a script. They can both be found in the Library Browser. Let us consider Example 22 described in Section 3.1.1.

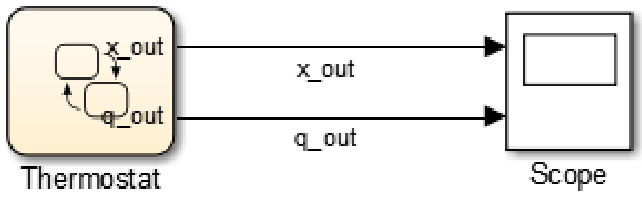


Figure B.1: Chart and scope objects within Simulink of the Example 22 in Section 3.1.1.

Within the chart object, we declare state variables in a state element which corresponds to a hybrid automaton. We can name the locations, declare the initial condition, vector fields, guards and reset functions. For the example, we constructed two locations, On and Off. In the state element, the invariants are defined by the guards from the origin location to the destination one. The model does not support the matrix/vector notation. The derivative is represented by the variable plus suffix "*_dot*" and the output is represented by the variable plus suffix "*_out*". For this example the continuous variable is x for the temperature and the discrete variable is q for the location representation, 1 for location On and 0 for location Off.

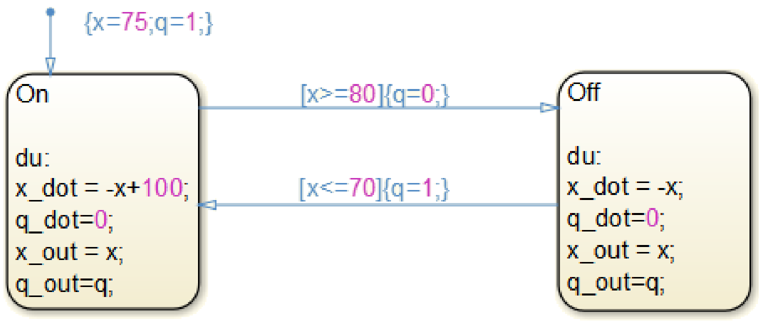
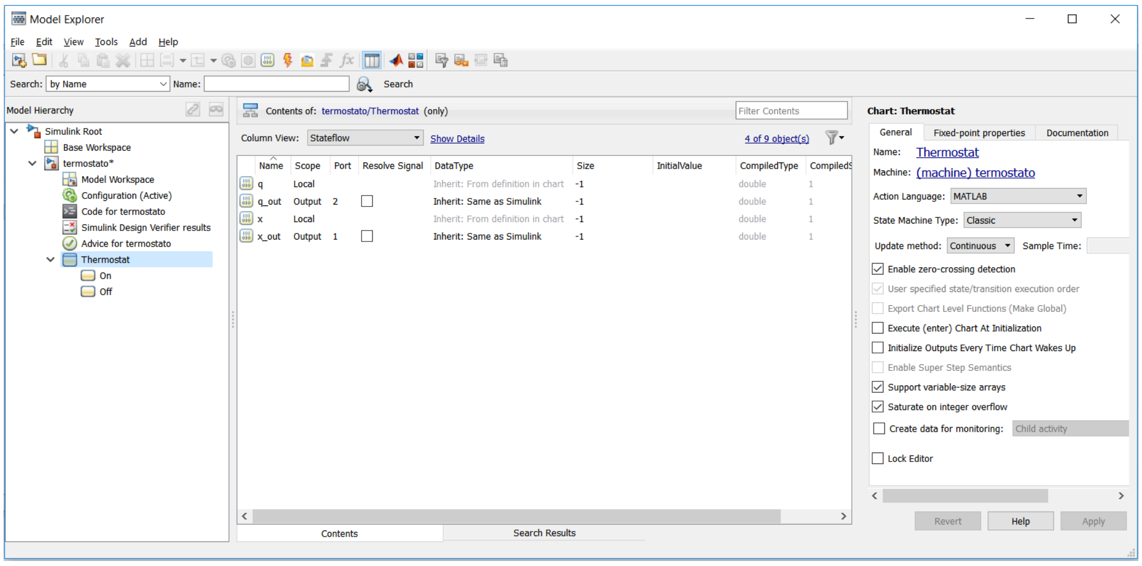


Figure B.2: State elements representing locations within the Chart object.

When a continuous variable is created, automatically its derivation is created. We create continuous variables and outputs in the Model Explorer by adding "*Data*" in the menu bar. The thermostat model is a classic state machine with a continuous update method that allows derivative declaration.



Contents of: termostato/Thermostat (only)

Column View: Stateflow   Show Details   4 of 9 object(s)

	Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledType	CompiledS
	q	Local			Inherit: From definition in chart	-1		double	1
	q_out	Output	2	☐	Inherit: Same as Simulink	-1		double	1
	x	Local			Inherit: From definition in chart	-1		double	1
	x_out	Output	1	☐	Inherit: Same as Simulink	-1		double	1

Chart: Thermostat

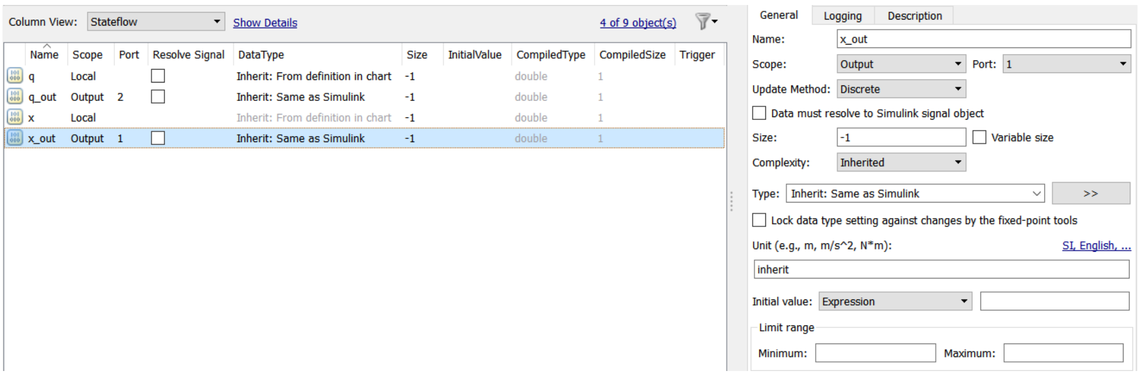
General | Fixed-point properties | Documentation

Name: Thermostat
Machine: (machine) termostato
Action Language: MATLAB
State Machine Type: Classic
Update method: Continuous   Sample Time:

- ✓ Enable zero-crossing detection
- ✓ User specified state/transition execution order
- ☐ Export Chart Level Functions (Make Global)
- ☐ Execute (enter) Chart At Initialization
- ☐ Initialize Outputs Every Time Chart Wakes Up
- ☐ Enable Super Step Semantics
- ✓ Support variable-size arrays
- ✓ Saturate on integer overflow
- ☐ Create data for monitoring: Child activity
- ☐ Lock Editor

Figure B.3: Model Explorer - chart configuration.

The output variables and the discrete variable "*q*" are configured as discrete update method and continuous variable "*x*" is configured as continuous update method.



Column View: Stateflow Show Details 4 of 9 object(s)

	Name	Scope	Port	Resolve Signal	DataType	Size	InitialValue	CompiledType	CompiledSize	Trigger
	q	Local		☐	Inherit: From definition in chart	-1		double	1	
	q_out	Output	2	☐	Inherit: Same as Simulink	-1		double	1	
	x	Local			Inherit: From definition in chart	-1		double	1	
	x_out	Output	1	☐	Inherit: Same as Simulink	-1		double	1	

General Logging Description

Name: x_out

Scope: Output Port: 1

Update Method: Discrete

☐ Data must resolve to Simulink signal object

Size: -1 ☐ Variable size

Complexity: Inherited

Type: Inherit: Same as Simulink >>

☐ Lock data type setting against changes by the fixed-point tools

Unit (e.g., m, m/s^2, N*m): SI, English, ...

inherit

Initial value: Expression

Limit range

Minimum: Maximum:

Figure B.4: Model Explorer - variables configuration.

One advantage of this tool is that data management in Matlab is resourceful. We can plot multiple trajectories in one graph, which is an easy way to compare data. This kind of comparison can not be done in SpaceEx, only two variables can be plotted by graph. However, event implementation in this tool is not successful. Transitions can only occur via guard enabling.