State of art

"Analysis and robust control of discrete events process by non-deterministic model and set methods : application in Cartesian varying maze resolution, with uncertain dynamics or measures."

Authors :

Johanne Bakalara
Claire Paya
Alexandre Armengaud
Lionel Mery
Lucien Rakotomalala
Ibrahim Atto
David Tocaven

Supervisor :
Yann Labit
Julien Vanderstraeten

Client :
Sylvain Durola

Problematic

The project focuses on the modeling and analysis with Discrete Events Systems (DES) of a system composed of a dynamic labyrinth and two objects moving inside the labyrinth.
This analysis will be used to identify properties to test this uncertain model. We will contextualize our project with current knowledge in DES modeling.
First, we will redefine the definition of what we will call a "labyrinth" and then we will recall the formal definitions of DES [7] [8].
In a second step, we will discuss more precisely the properties and products that we will use for the analysis and verification of our models. We will use the formal definitions explained in these books [2] [3].
The last part will remind us the scientific approach chosen and the types of verification and validation that will allow us to test our prototypes.

January 23, 2018

Contents

1 | Maze Definition

Our subject focuses on the modeling and analysis of a labyrinth, so we first presente a formal definition of a labyrinth throughout this state of art.
We begin by using an existing definition: ***"A maze is a grid-like two-dimensional area of any size, usually rectangular. A maze consists of cells. A cell is an elementary maze item, a formally bounded space, interpreted as a single site. The maze may contain different obstacles in any quantity."*** from Foundations of Learning Classifier Systems by Larry BULL and T. KOVACS.

In this definition, we can see the concept of dimension is important, it is introducing the notion of a cell. We chose to define a maze as a set of cells where the number of cells gives the dimension of a maze. This set of cell is confronting to all the obstacles that determines the possible and impossible paths.
In our study we are developing a maze with dynamics walls, an object trying to escape and a second one who is trying to find the first object. Each object choose a cell, he can go to the cell above, below, on the left or on the right depending on walls around him (they cannot cross the walls). So, a cell surrounded by four walls is not reachable and if an object is on, he is trapped.

Now let us try to characterize our maze still based on paper from Anthony J. Bagnall and Zhanna V. Zatuchna listed in the previous book.

Size: 2D rectangular with 25 cells (5x5)

Distance from "Escape": uncertain (short, medium, long)

Obstacles: moving walls, enemy (one object), edges of the maze

Type of objects: Object, Walls

Maze Dynamics: We have dynamics walls so our maze is called as *dynamic maze*. A cell is reachable if there is no wall behind. If an object is on a cell, she is not reachable anymore by an other object.

2 System

This part is an introduction to the concept of system in order to connect it to a labyrinth. We will find a complete definition obtained from scientist organizations and from our own meaning of system.

2.1 Definition

The objective of this section is to give our definition of a system. The concept of system is explained with several primitives from multiple scientific and engineering organizations. We chose two of them :

- "An aggregation or assemblage of things so combined by nature or man as to form an integral or complex whole" from Encyclopedia Americana .
- "A combination of components that act together to perform a function not possible with any of the individual parts" from IEEE Standard Dictionary of Electrical and Electronic Terms

We chose to keep both definitions for two features : association of elements and the concept of function of a system. Therefore, for our next presentation, we will define a system as a collection of components which are interrelated in an organized way, and these associations work together for the accomplishment of some logical and purposeful ends.

2.2 Modeling of System

Our initial approach for system modeling will be defined in this section. First, to enable analysis and command of our system, we have to respect the concept of input/output. In that way, it will be possible to describe mathematical behavior of our system by defining the outputs in function of input. To this end, we initiate a set of *input variables* and *output variables* [2], respectively $u(t)$ and $y(t)$, two vectors sizable $1 \times p$ and $1 \times m$:

$$u(t) = \begin{pmatrix} u_1(t) & ... & u_p(t) \end{pmatrix}^T \text{ for } t \in [t_0, t_f]$$
$$y(t) = \begin{pmatrix} y_1(t) & ... & y_p(t) \end{pmatrix}^T \text{ for } t \in [t_0, t_f]$$

To enable a better representation of the system, we will add our modeling system the concept of state. This term consist in representing a set of variables named x, the present state of a system in $t \leq t_0$ and it will be used to calculate the set of outputs in t and a next representation of this state, in $t + \tau$. This state vector is as defined below:

$$x(t) = \begin{pmatrix} x_1(t) & ... & x_n(t) \end{pmatrix}^T \text{ for } t \in [t_0, t_f]$$

In addition, we would like to join the *inputs*, *outputs* and *state variables* together that lead us to the concept of *state space* (sections 1.2.6 of [2]). Here are relationships between u, y and t if the vector of *state variables* is correctly chosen. It refers to the *dynamics* of the system and it can take a few possible representations, but we chose to present you this one, it seems to us to be the most complete one:

$$\begin{cases} \dot{x}(t) = f(x(t), \delta, u(t), t) \\ y(t) = h(x(t), \delta, u(t), t) \end{cases} \tag{2.1}$$

The first function f represents the system dynamics using a differential equation of the system state variable. The second function g models the output exigences. We have modeled various parameters thanks to this equation. Here is the list :

- $\delta \in \Delta$ incertitude of the system, we chose not to forget the potentially unknown and instable parameters of the system.
- $t \in \mathbb{R}^+$ evolution of time of some parameters.

These types of system modeling are called nonlinear, variant and non deterministic system with continuous time. They are not suitable for mathematical analysis (although possible) or control. Assumptions are then imposed to remove non-linearity, variability and non-determinism through computational steps so as not to stray too far from the complex model.

Many methods exists to simplify this complex model; they will not be detailed here, we will see later that the theoretical assumptions on the models will enough. We just have to remember that each hypothesis added on the complex model yields:

- Linearization gives a model defined by:

$$\begin{cases} \dot{x}(t) = A(t,\delta)x(t) + B(\delta,t)u(t) \\ y(t) = C(t,\delta)x(t) + D(\delta,t)u(t) \end{cases} \tag{2.2}$$

  with the linearization of f and g on two linear composition A and B for f and C and D for g.

- Invariant prevents the model from being time-dependent:

$$\begin{cases} \dot{x}(t) = f(x(t),\delta,u(t)) \\ y(t) = g(x(t),\delta,u(t)) \end{cases} \tag{2.3}$$

  Here we simplify the elements that were based on time, i.e. it could change instantly. This hypothesis means the following model is invariant.

- Deterministic We associate the uncertainties within the system with some elements of the system:

$$\begin{cases} \dot{x}(t) = f(x(t),u(t),t) \\ y(t) = g(x(t),u(t),t) \end{cases} \tag{2.4}$$

  Uncertain parameters are now associated with reliable parameters, we have an approximation of our model: it is named deterministic.

Now we propose this assumption, let's look at the shape of the state space model. This one is described below, and called *Deterministic Continuous Time Invariant Linear System*:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{2.5}$$

In the next step, we will see a subclass of this type of systems, where continuous time is not an evolutionary variable anymore. System-class called Discrete Time System exists. The evolution is based on a sample sequence kTe with $k \in \mathbb{N}$ the sample number and $Te \in \mathbb{R}$ a sample frequency and another sub-class, the Discrete Event System, where the sampling frequencies only dependent upon events which are now presented.

3 Discrete Events Systems

As we introduce a definition of system and define hypothesis about it, we can now focus on a separated part of sequential system : the Discrete Event Systems also named DES. They have been introduced by B.ZEIGLER and have been evolving ever since.

3.1 Definition of Discrete Event systems

Discrete Event Systems (DES) are dynamic systems evolving at the rate of a series of discrete events. Discrete events are events that can possibly be physical or numerical, and they are over varying intervals, sometimes unknown [7]. These physical events are associated with occurrences in our system, and they will be developed later in this chapter.

Therefore, DES depend on a set of *state variables*. We will take the previous vector to adapt it to the DES. In this new case, the *state variable* x is the set of states attainable by the system. We note this :

$$x \in \mathcal{Q} \text{ with } \mathcal{Q} \text{ ensemble of attainable states}$$

Attainable states are most of time an infinite number, especially if the assumptions posed in 2.2 are respected.

The inputs of the system that was previously u rated will now belong to the events set. The dynamics of the system no longer depend upon a time, but on all possible events. Thanks to that new informations, we obtain a mathematical relationship that we will simplify by the following formula :

$$\begin{cases} x_s = f(x,u) \\ y = g(X,u) \end{cases} \tag{3.1}$$

the two functions f and g being linear, non deterministic and invariant. The dynamic system is controlled by f : it uses event on u and the present state x in order to evaluate the new system state x_s.

If for each inputs, we linked it with an event, we allowed to obtain a set of events rated Σ. We could find, for example, a formulation of DES which is well known and which could be used to study the diagnosability of a system [8]:

$$G = (x, \Sigma, \delta, x_0) \tag{3.2}$$

The f function presented in 3.1 is now represented by δ, the transition function. Then, we have x_0 is the initial state, Σ is the set of events and x the actual state.

3.2 DES and finite state

A part of DES exists with a specific state vector : it is not a vector and can only take one value. We called it a finite state Discrete Event System because :

$$x \in \mathcal{Q} \text{ where } Q \text{ is only between [0,n]}$$

These DES admit another theory that we would like to introduce right now : the theory of automatons and languages. This finite state DES is a logical model as presented by Ramadge and Murray Wonham [7], and it is possible to list the occurrences of events in this type of system.

In the following sections, we will introduce our labyrinth problem in the DES concept then, we will study more precisely the theory just such introducing in this chapter.

3.3 Introduction of DES in labyrinth

In order to introduce DES in our dynamic labyrinth, we join each movement of object or wall as an event. So we have a list of elements :

$$\Sigma = \{H, B, G, D, W_d, W_r\}$$

with each event equivalent to :

- H for up movement
- B for down movement
- G for left movement
- D for right movement
- W_d for wall down movement
- W_r for wall right movement

Here is a quick representation of a possible sequence of a labyrinth in 2D, to this 9 possible cells labyrinth. Each movement of an object is subject to the system dynamics indicating it can only move from cell to cell.

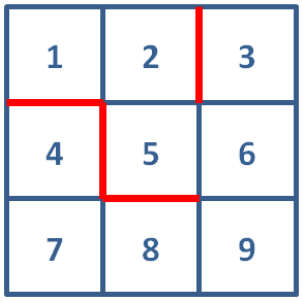


Figure 3.1: Example - Labyrinth

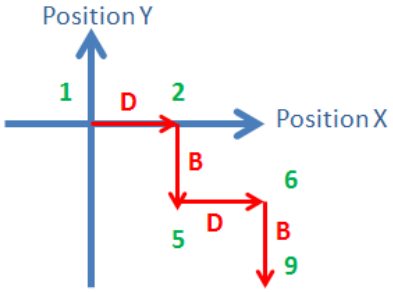


Figure 3.2: Example - Graph of path

The sequence represented here starts in the state modeled on the figure 3.1, object starts in position 1. The sequence going from 1 to 9 is the one shown in 3.2 : $D \rightarrow B \rightarrow D \rightarrow B$.

In the next part, we will explain the tools of automatons theory that we will use for the continuation of the project to analyze his kind of sequence and we will define a strong formalism we are going to use.

4 Automaton

4.1 Definition and Representation of an Automaton

A finite automaton is a Finite State System composed of discrete inputs and outputs. It is a mathematical model of system composed of a finite number of states. Each state summarize informations and the configuration of these states determine the behavior of the system. We could for example model a human brain with a finite state system considering that each neuron contains informations which can be described by a small number of bits. Furthermore the number of neurons is limited to 2^{35} at most.
We consider an automaton as a set of states and transitions from state to state that occur on input symbols. We can model the system as a graph, called a *transition diagram* which is associated with a finite automaton. The verticals of the graph correspond to the states of the finite automaton. Each state is modeled by an unique circle labeled and transitions are modeled by an arrow from one state to another one that is also labeled. An event can occur many times but a state is unique.
This definition is inspired by the book *introduction Automata Theory Langage and Computation* written by John E Hopcroft, R.Motwani and Jeffrey D.Ullman. [3]

For example : A light

We model a light that starts and switch on off if one press the button :

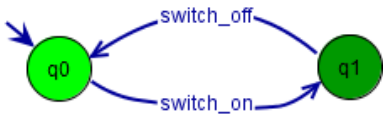


Figure 4.1: Example 1 - finite automaton

This automaton is composed of :

- two states : q0,q1
- an initial state : q0
- two transitions : switch_off, switch_on

In a maze context, objects and the dynamics walls have a behavior we can model with automaton. The object moves can have as possible transitions *up*, *down*, *left*, *right*, the walls can also have moves, *up*, *down*, *left*, *right*. Each cell of our maze is a state where few events can occur. For example, each possible event that can happen on every cell - if we don't know the position of the object - is described by the next automaton 4.3 :

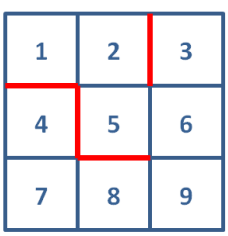


Figure 4.2: Example - Maze

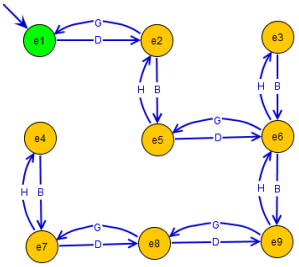


Figure 4.3: Example - Automaton

4.2 Formal representation of an deterministic automaton

We formally denote a finite automaton by the next 5-tuple :

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F) \tag{4.1}$$

With :

Q	:	finite set of states.
Σ	:	finite input alphabet.
q_0	:	$q_0 \in Q$ the initial state.
F	:	$F \subset Q$ the set of final states
δ	:	the transition function mapping Q x Σ to Q

Alphabet : set of all events known by the automaton

String : one sequence of transitions

The transition function : a string x is accepted by the automaton $\mathcal{A}$ if $\delta(q_0, x) = p$

4.3 Formal representation of a nondeterministic automaton

As explained by Hopcroft, Motwani and Ullman [3], *"a nondeterministic automaton has the power to be in several states at once"*. In the formalization we supposed that a same input can arrive to several states. The definition of an automaton remains mainly the same.

$$\mathcal{A} = (Q, \Sigma, \delta, q_0, F) \tag{4.2}$$

With :

Q	:	finite set of states.
Σ	:	finite set of input alphabet.
q_0	:	$q_0 \in Q$ the initial state.
F	:	$F \subset Q$ the set of final states
δ	:	the transition function mapping Q x Σ to $\mathcal{P}(Q)$

The difference between a nondeterministic and a deterministic automaton is the type of value that δ returns. Thus, the transition function is extended :

$$w = xa$$

where a is the final letter of w and x is the rest of w

$$\widehat{\delta}(q, x) = \{p_1, p_2, ..., p_k\}$$

$$\bigcup_{k}^{i=1} \delta(p_i, a) = \{r_1, r_2, ..., r_m\}$$

Then

$$\widehat{\delta}(q, w) = \{r_1, r_2, ..., r_m\}$$

Cassandras define nondeterministic finite automata as [2]:

$$\widehat{\delta} : Q \times \Sigma \to 2^X$$

$$\widehat{\delta}(x, s) \subseteq Q$$

with

$$q_0 \subseteq Q.$$

4.3.1 Language Notation and Definitions

Definition (Language and notation, [2]) : A language defined over an event set L is a set of finite-lenght strings formed from events in E.
As an example, let $\Sigma = \{a, b, g\}$ be the set of events. We define a language : $L = \{\epsilon, a, abb\}$.
The definition of the language for an automaton is really useful to analyze it. Language is a mathematical tool used to do operations, in order to extract many properties for automata. We recall how to express the language of an automaton then the operations below.

4.3.2 Languages Represented by Automata

Definition by Cassandras[2] :
The language generated by $A = (Q, \Sigma, \delta, q_0, F)$ is :

$$\mathcal{L}(A) := \{s \in \Sigma^* \delta(q_0, s)[is\ defined]\}$$

The language marked by A is :

$$\mathcal{L}_m(A) := \{s \in \mathcal{L}(A) : \delta(q_0, s) \in F\} \tag{4.3}$$

with the extended transition function : $\delta : Q \times \Sigma^* \to F$ with $\epsilon \in \mathcal{L}(A)$

The language generated by A represents all strings existing in the automaton starting at the initial state. The string is the concatenation of the event labels of the transitions for a path in an automaton. It is also called a sequence.
The marked language represents all strings ending at a marked state. We also said that the language is *recognized* by the automaton.
For instance :

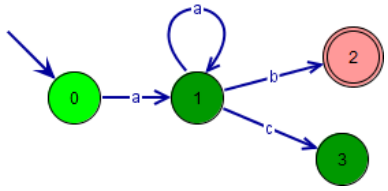


Figure 4.4: Automaton example

This finite state automaton is $\mathcal{A} = (Q, \Sigma, \delta, q_0, F)$ with :

- $Q = \{0, 1\}$
- $\sigma = \{a, b, c\}$
- $q_0 = 0$
- $F = \{2\}$
- $\delta(0, a) = 1$, $\delta(1, a) = 1$, $\delta(1, b) = 2$, $\delta(1, c) = 3$

We can see from the figure and the definitions that :

- $\mathcal{L}_m(A) = aa * b$
- $\mathcal{L}_g(A) = aa * (b + c)$

We formally define the language for a deterministic automaton as :

$$L(\mathcal{A}) = \{x | \delta(q_0, x) \cap F\}$$

if a state is accepted by the one finite automaton, it is a *regular set*

The language accepted by the nondeterministic finite automaton is defined by :

$$L(\mathcal{A}) = \{w | \widehat{\delta}(q_0, x)\} \cap F \neq \emptyset$$

If a main sequence of choices from a start state to any accepting state if possible, it accepts a string w .

4.3.3 Operations

Let's consider two languages L and M and an alphabet E.

Union : $L \cup M$ set of strings which are in the languages L or M.

Concatenation : the concatenation of L and M is detoned by a "dot" or nothing as $L.M$ or LM. LM contains each string in L concatenated with each string in M. Each string must only appear once. The empty string ϵ is the identity element : $u\epsilon = \epsilon u = u$

$$L, M \subseteq E^*$$

$$LM := \{s \in E^* : (s = s_L s_M) and (s_L \in La) and (s_M \in L_M)\}$$

Kleene-closure/ Kleene star : it is denoted L^*, any number of possible strings with repetitions, concatenating all, including the empty string ϵ.

$$L \subseteq E^*$$

$$L^* := \in \cup L \cup LL \cup LLL \cup ...$$

Projection :

$$L \subseteq E_l^*$$

$$P(L) := \{t \in E_s^* : (\exists \in L)[P(s) = t]\}$$

$$L \subseteq E_s^*$$

$$P^{-1}(L_s) := \{s \in E_l^* : (\exists t \in L_s)[P(s) = t]\}$$

Example of operations for Languages : $E = \{a, b\}$ and $L = \{c, acb, abc, cacacb, aacbcab\}$

Consider the projection : $P : E_l^* \to E^*$

$$P(L) = \{\epsilon, ab, aab, aabab\}$$
$$P^{-1}(\{\epsilon\}) = \{c\}^*$$
$$P^{-1}(\{b\}) = \{c\}^* bc*$$

4.3.4 Properties

The properties below were extracted by the operations.

Language-equivalent automata [2] : Automata G_1 and G_2 are said to be language-equivalent if $\mathcal{L}(G_1) = \mathcal{L}(G_2)$ and $\mathcal{L}_m(G_1) = \mathcal{L}_m(G_2)$

Blocking : Automaton G is said to be blocking if $\mathcal{L}_m(G_1) \subset \mathcal{L}(G)$ and nonblocking when $\mathcal{L}_m(G_1) \subset \mathcal{L}(G)$

Accessibility : A state is said *accessible* a sequence/word exists from the initial state to this state. With the definition $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$ explained above, we can apply them to find all existing words from the initial state to all reachable states :

$$A_{ac} = \{q \in Q | (\exists s \in \Sigma^*)[\delta(q_0, s) = q]\}$$

An automaton is defined as *accessible* if all its states are accessible : $Ac(A) = Q$.
Despite as it is explained in [2] by *Cassandras and Lafortune*, we will consider that an accessible state is not necessarily marked by $\mathcal{A}$.

Coaccessibility : A state is named *coaccessible* a sequence/word it exists from this state to a marked state. With the definition $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$ explained above we can apply them in order to find all existing words from all states to one of the marked states :

$$C_o(A) = \{q_j \in Q | \exists s \in \Sigma^* [avec] \delta(q_j, s) \in F\}$$

An automaton is defined as *coaccessible* if all its states are coaccessible : $C_o(A) = Q$. Thus, $\mathcal{L}(G) = \overline{\mathcal{L}_m(G)}$ Coaccessibility is closed to the concept of blocking : an automaton is *blocked* if $\mathcal{L}(G) \neq \overline{\mathcal{L}_m(G)}$. It means, states do exist that are accessible but not coaccessible.

4.3.5 Composition Operations

Two types of operations for deterministic automata exist : product and parallel composition. They can be directly be applied in classical methods on the graph but it comes from the language operations.
Product composition between two automata lets show the synchronization on a common event. An event can occur if it is in both Automata but it also contains the independent event of each automaton. We formally defined the product of G_1 and G_2 as the automaton :

$$G_1 \times G_2 := A_c(Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, Q_{m1} \times Q_{m2})$$

$$\mathcal{L}(G_1 \times G_2) = \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$$

$$\mathcal{L}_m(G_1 \times G_2) = \mathcal{L}_m(G_1) \cup \mathcal{L}_m(G_2)$$

In a *parallel composition* an event can only occur if both automata execute it simultaneously : they are synchronized. The parallel composition of G_1 and G_2 is the automaton :

$$G_1 \parallel G_2 := Ac(Q_1 \times Q_2, \Sigma_1 \cup \Sigma_2, \delta, F_1 \times F_2)$$

$$\mathcal{L}(G1 \parallel G2) = P_1^{-1}[\mathcal{L}(G1)] \cap P_2^{-1}[\mathcal{L}(G2)]$$

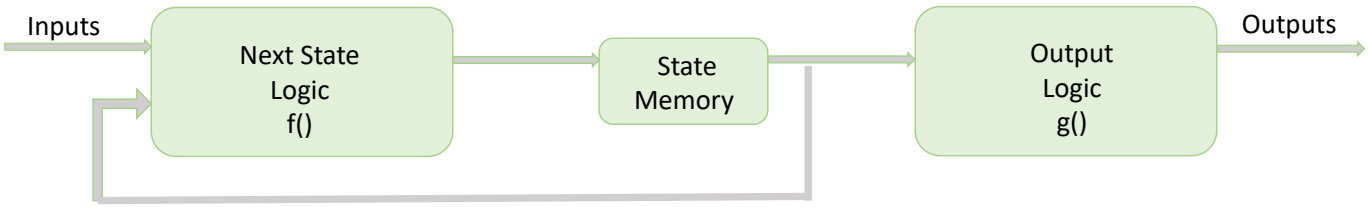
$$\mathcal{L}_m(G1 \parallel G2) = P_1^{-1}[L_m(G1)] \cap [P_2^{-1} L_m(G2)]$$

4.3.6 Modeling a system with Automata - Inspired by the Supervisory Command

The model that describes all existing situations of a system and its behavior (events, inputs, outputs, states) is called *The process model.* If we want to choose a specific behavior of this system, we have to create a a model that describes these objectives. If we need to reach these objectives (for instance : "arrive in a marked state "), we must to create a model that describes these objectives. It is usually called *the objectives command.*
The *Composition Operation* of both models describes the appropriate behavior because it only contains the common events from both models. This new model is called the *Command model.*
In our context, all events are observable and controllable, so Cassandras explains the usefulness for using the parallel composition or the product :
The choice of product or parallel composition to compose [the specification model] called H_{spec} with [the process Model called] G is based on the events that appear in the transition diagram of H_{spec} and on how we wish to define the event set of H_{spec} :

- *If the events that can be executed in G but do not appear in the transition diagram of H_{spec} are irrelevant to the specification that H_{spec} implements, then we use parallel composition and define the event set of H_{spec} to be those events that appear in its transition diagram.*
- *On the other hand, if the events that can be executed in G but do not appear in the transition diagram of H_{spec} are absent from H_{spec} because they should not happen in the admissible behavior L_a, then product is the right composition operation.*

Equivalently, parallel composition can still be used provided the event set of H_{spec} is defined carefully and includes the events that should be permanently disabled.
In most cases, all the states of H_{spec} will be marked so that marking in [the result of the operation] H_a the will be solely determined by G.



4.4 Implementation by sequential system blocks

The automota can be implemented by a sequential system because the evolution of the system depends on the present state, the next state but also of inputs and outputs of the system. The implementation based on a Moore Machine is a formal way to implement automota.

We define the next state with our transition function, we memorize the present state and generate the outputs and then we define the new next state with the present state and the new inputs.
It's adapted to be implemented in different languages. Our project uses the language object Matlab implementation, where each class has a present State containing his intern situation and his own evolution functions $f()$, $m()$ and $g()$ that read the inputs, calculate outputs and the next State.

5 Verification and validation

5.1 Approach methods

We chose to study three types of approaches. There are adapted for researches and development fundamental or applied. During our project we shall use the next three approaches by using theoretical proofs or validation tests. This part consists of introducing these different types of reasoning.

5.1.1 Inductive Approach

An inductive approach begins by observing of our system used to generalize our theories.[Wilson, J. (2010) "Essentials of Business Research: A Guide to Doing Your Research Project" SAGE Publications, p7] [4]
This reasoning is composed of four steps :

- Observing
- Analyzing the observations
- Generalizing thanks to hypothesis
- Verifying hypothesis

5.1.2 Deductive Approach

The deductive reasoning consists in testing a theory then confirming if throw the analysis of the observations [Wilson, J. (2010) "Essentials of Business Research: A Guide to Doing Your Research Project" SAGE Publications, p5] [4]
We can prove and explain implied a cause to effect relationships. This reasoning is composed of four steps :

- Deducing hypothesis from theory
- Testing hypothesis
- Analyzing results
- Modifying theory if necessary

5.1.3 Abductive Approach

The abductiv reasoning consists in finding explanations of an experiment and hypothesis. It is used for scientific discoveries because it is more convenient to innovative approaches. As explained by Wilson, the abductiv approach contains four steps :

- Recognizing the existence of an abductiv problem
- Identifying solutions
- Finding possible solutions
- Explaining solutions and implementing them to solve the problem

5.2 Test method

In order to validate the code, it is necessary to set up a method of verification and validation of the code, it is notably possible to use *oracles*. An oracle is a small code that runs along the code. It is used to verify the success or failure of a test. It means, we should determine if the result obtained at the end of the test corresponds to the expected one [1]. Several types of tests are needed, it depends on an expected scenarios. These types of tests correspond to data tests that must be sent at the oracle input. At the output of the oracle, the result will confirm the validity or invalidity of the tested code. These test cases can either have an inductive approach by performing a multiply tests or an abductive approach by performing precise and targeted tests for each critical case.

Bibliography

[1] R. Carver and Yu Lei. Stateless techniques for generating global and local test oracles for message-passing concurrent programs. *Journal of Systems and Software*, 136:237 – 265, 2018.

[2] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. SpringerLink Engineering. Springer US, 2009.

[3] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Pearson Addison-Wesley, 2007.

[4] J.Wilson. *Essentials of Business Research: A Guide to Doing Your Research Project*. 2010.

[5] W.S. Levine. *The Control Handbook*. Electrical Engineering Handbook. Taylor & Francis, 1996.

[6] Zina O'Leary. *The Essential Guide to Doing Research*. SAGE, 2004.

[7] P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, Jan 1989.

[8] M. Sampath, R. Sengupta, S. Lafortune, K. Sinnamohideen, and D. Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1556, Sep 1995.

[9] Fernando R. Velázquez-Quesada, Fernando Soler-Toscano, and Ángel Nepomuceno-Fernández. An epistemic and dynamic approach to abductive reasoning: Abductive problem and abductive solution. *Journal of Applied Logic*, 11(4):505 – 522, 2013. Combining Probability and Logic: Papers from Progic 2011.